University of California
Santa Barbara

Reinforcement Learning for Generative Art

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Media Arts & Technology

by

Jieliang Luo

Committee in charge:

Professor George Legrady, Chair
Professor Tobias Höllerer
Professor Yon Visell
Dr. Hui Li

March 2020

The Dissertation of Jieliang Luo is approved.

Professor Tobias Höllerer

Professor Yon Visell

Dr. Hui Li

Professor George Legrady, Committee Chair

March 2020

Reinforcement Learning for Generative Art

by

Jieliang Luo

Acknowledgements

I would love to thank my dissertation committee for their supports, guidance, understanding, and inspirations throughout my time at this program. Foremost, I thank Prof. George Legrady for his consistent supports and encouragements to develop my works, and inspiring me to explore the fields of media art. I appreciate Professor Tobias Höllerer for his ongoing support of my work, raising the motivations, and understanding. I appreciate Prof. Yon Visell for his devoted help, motivations, and inspiring discussions to develop my dissertation. I appreciate Dr. Hui Li for her supports, understanding, and inspiring discussions. I would also like to thank the whole MAT community as well. I acknowledge financial support from the Media Arts and Technology graduate program and Robert W. Deutsch Foundation.

Curriculum Vitæ

Jieliang Luo

Education

09.2014-01.2020 **PhD in Media Arts and Technology**
University of California, Santa Barbara
Dissertation Bridging Reinforcement Learning & Generative Art
Advisor: Prof. George Legrady

09.2010-03.2013 **MA in Emergent Digital Practice**
University of Denver
Advisor: Prof. Christopher Coleman

09.2006-06.2010 **BE in Digital Media Technology**
Beijing University of Posts & Telecommunications

Work Experience

Begins at 02.2020 **Autodesk Research** (San Francisco, CA)
Principal AI Researcher

06.2018-12.2019 **Autodesk Research** (San Francisco, CA)
Machine Learning & Robotics Intern

- Published "Dynamic Experience Replay", a reinforcement learning technique to enhance robotic assembly, at Conference of Robot Learning (CoRL) 2019.
- Developed a distributed reinforcement learning framework for architectural-scale robotic assembly, featuring data-sampling enhancement and sim-to-real transition.

01.2015-06.2018 **Experimental Visualization Lab, UCSB** (Santa Barbara, CA)
Graduate Researcher & Lecturer

- Designed and developed an autonomous multiple robotic-camera system for studying distributed reinforcement learning on embedded systems. A version of the system named "Machinery Interference" was featured at Beijing Times Arts Museum.
- Developed an interactive fluid simulation called "Anamorphic Fluid", featured in USA, Thailand, and South Korea.
- Taught two courses as lecturer
  - Data Visualization (Processing and MySQL)
  - Mobile Robotics (Raspberry Pi, Arduino, and OpenCV)

09.2017-06.2018 **Koc Lab, UCSB** (Santa Barbara, CA)
Graduate Researcher

- Published a book chapter called "Reinforcement Learning and Trustworthy Autonomy" at Cyber-Physical Systems and Security.
- Developed a toolkit for visually understanding deep reinforcement learning policies.

06.2015-09.2015 **InTouch Health** (Santa Barbara, CA)
Software Engineer

- Developed middleware and several prototypes for robot-human interaction, task allocation, control, and path planning

Publications

2019

Jieliang Luo and Hui Li, "Dynamic Experience Replay", 2019 Conference of Robot Learning (CoRL)

Yongxiang Fan, **Jieliang Luo**, and Masayoshi Tomizuka, "A Learning Framework for High Precision Industrial Assembly", 2019 International Conference on Robotics and Automation (ICRA), 811-817

2018

Jieliang Luo and Sam Green, "Bridging Reinforcement Learning and Creativity: Implementing Reinforcement Learning in Processing", SIGGRAPH Asia 2018 Courses, 3

Jieliang Luo, Sam Green, Peter Feghali, George Legrady, and etin Kaya Koc, "Reinforcement Learning and Trustworthy Autonomy", Cyber-Physical Systems and Security, 191-217

2017

Jieliang Luo, Donghao Ren, and George Legrady, "Anamorphic Fluid: Exploring Spatial Organization and Movements of Images in A Simulated Fluid Environment", Proceedings of The 10th International Symposium on Visual Information Communication and Interaction, 63-64

Exhibitions

11.2019 **Beyond the Body**, "LAVIN", SWISSNEX, San Francisco, CA, USA

08.2019 **ACM SIGGRAPH 2019**, "LAVIN", Los Angeles Convention Center, Los Angeles, CA, USA

07.2019 **ChinaVIS 2019**, "Human Portraits Decompositions 2.0", Longemont Hotels, Chengdu, Sichuan, China

10.2018 **IEEE VIS 2018**, "Human Portraits Decompositions", Estrel Hotel & Congress Center, Berlin, Germany

08.2018	**XcelerAtor**, "Machinery Interference", Times Arts Museum, Beijing, China
11.2017	**SIGGRAPH Asia 2017**, "Abstract Reality", Bangkok International Trade & Exhibition Center, Bangkok, Thailand
06.2017	**Currents New Media Festival**, "Anamorphic Fluid II", Santa Fe, NM, USA
08.2016	**SV+VS**, "Anamorphic Fluid", Dongdaemun Design Plaza Museum, Seoul, South Korea
04.2016	**Art & Science**, "Voro Space", California Nano Systems Institute, Santa Barbara, CA, USA
01.2016	**SV+VS**, "Anamorphic Fluid", Fellows of Contemporary Art, Los Angeles, CA, USA
11.2015	**Day & Night**, "Anamorphic Fluid", Edward Cella Art & Architecture, Los Angeles, CA, USA
11.2012	"Touch History", Holocaust Memorial Social Action Site, Denver, CO, USA

Awards

2018	**Individualized Professional Skills Grant**, UC Santa Barbara
2017	**International Student Award**, UC Santa Barbara
2017	**Graduate Fellowship**, UC Santa Barbara
2016	**Robert W. Deutsch Foundation Fellowship**, Deutsch Foundation
2016	**Integrative Graduate Education and Research Traineeship Fellowship**, UC Santa Barbara
2015	**The University Student Aid Program Fellowship**, UC Santa Barbara
2012	**Graduate Assistant Scholarship**, University of Denver
2011	**Deans Scholarship**, University of Denver

Abstract

Reinforcement Learning for Generative Art

by

Jieliang Luo

Reinforcement learning (RL) is an efficient class of sequential decision-making algorithms that have achieved remarkable success in a broad range of applications, such as robotic manipulations, strategic games, or autonomous driving. The most well-known example of reinforcement learning is AlphaGo, a computer program that plays the board game Go and outperforms top human Go players. Unlike other two major machine learning categories, supervised learning and unsupervised learning, in which media artists are actively engaged, reinforcement learning has yet to result in many creative applications.

Generative art is usually driven, in whole or in part, by autonomous systems that are derived from a set of rules. Interestingly, an RL policy can be seen as an autonomous system where the rules are learned by interacting with its environment. Regardless of its initial purpose, reinforcement learning has the potential to expand the boundary of generative art. However, a formal process of applying reinforcement learning to generative art does not yet exist and the current RL tools require an in-depth understanding of RL concepts.

To bridge the gap, the first part of the dissertation introduces a conceptual framework to adapt reinforcement learning for generative art. The framework proposes a term *RL-based generative art* to denote a novel form of generative art of which the use of RL agents is the key element. The creative process of RL-based generative art and possible emergent behaviors are discussed in the framework. This leads to a discussion of several author's related practices on generative art, deep-learning art, and reinforcement

learning. Those practices are critical for understanding the conceptual and technical details of each component in order to construct the framework.

The second part introduces RL5, a JavaScript library for rapidly prototyping RL environments and training RL policies in web browsers. The library combines RL algorithms and RL environments into one framework and is fully compatible with p5.js. RL5 is developed with a particular focus on simplicity to favor (re)usability of RL algorithms and development of RL environments. Specifically, the library implemented three RL algorithms, Tabular Q-learning, REINFORCE, and DDPG, to cover all the three families of model-free RL, and nine RL environments that six of them address autonomous agents in steering behaviors, which can be used as building blocks for complex systems. Finally, the author demonstrates four different use cases of how to apply RL5 for pedagogical and creative applications.

Contents

Chapter 1

Introduction

1.1 Motivations

> *In the last year (2016), generative machine learning models (GANs, RL, etc) and machine creativity have gotten a lot of attention. At the same time there have been significant advances in generative models for media creation and for design.*
>
> - Neural Information Processing Systems Foundation (NeurIPS)

For decades, machine creativity has been an essential topic in artificial intelligence (AI) and attracted numerous artists to explore its full extent and push the boundaries. Back in the 1970s, artists like Harold Cohen started to investigate the use of machines for autonomous drawings. Cohen's iconic project AARON[1], a computer program creating original artistic physical images, has been used as an artistic equivalent of the Turing Test. Recently, artworks driven by deep-learning techniques have become a trending art movement since Google released *DeepDream* [2] in 2015. The project is usually considered as one of the first artistic applications of deep learning. Since then, generative

machine learning models, like GANs [3] or RNN [4], have been widely adapted for artistic expressions.

As a breakthrough technology of deep learning in 2015, deep reinforcement learning (RL), an efficient class of sequential decision-making algorithms, has achieved remarkable success in a broad range of applications, such as robotic manipulations, strategic games, and autonomous driving. The most well-known example of deep reinforcement learning is AlphaGo [5], the first computer program to defeat Go world champions. The game Go is known as the most challenging abstract strategy board game for artificial intelligence due to its complexity of 10 to the power of 170 possible board configurations. Standard AI methods could reach the level of human amateurs as the methods cannot handle all the possible configurations of Go. Deep reinforcement learning, however, can adapt to uncertainties in unknown environments, which enabled AlphaGo to make an arguably optimal decision to an unseen configuration.

However, unlike other generative machine learning models that have been actively practiced in art, reinforcement learning has resulted in very few artistic applications. On the other hand, reinforcement learning has the potential to expand the boundary of generative art as the use of an autonomous system is a key element for both of them. A generative artwork is usually driven, in whole or in part, by one or several autonomous systems that are derived from a set of explicit rules defined by artists. An RL policy is essentially an adaptive autonomous system where the rules are learned by trial-and-error of its agent exploring the environment. Given the connection and difference between an RL policy and a rule-based autonomous system, it's promising that adapting reinforcement learning for generative art should open up new opportunities.

One gap between reinforcement learning and generative art is missing a clear workflow addressing how reinforcement learning should be adapted for generative art. Other popular generative machine learning models usually take a group of curated data as

inputs and output a group of generated data for artistic explorations, whereas reinforcement learning usually collects training data from an environment while a training session takes place. Thus, working with reinforcement learning requires developers to define an RL environment, which is not straightforward as selecting training data. Meanwhile, an appropriate tool is missing for non-experts to rapidly explore reinforcement learning. Current RL tools primarily focus on developing and evaluating novel RL techniques and normally a successful RL application takes hours, days, or even months to training. On the other hand, machine learning tools designed for creativity barely provide RL modules. Thus, the primary motivation of this research is to investigate how reinforcement learning can expand the boundary of generative art and how to improve the accessibility of reinforcement learning for creatives.

1.2 Problem Statement

Deep learning techniques have been widely applied for artistic explorations. Given the steep learning curve of those techniques, many attempts have been made to improve the accessibility of deep learning for creators. In general, deep learning can be categorized as supervised learning, unsupervised learning, and reinforcement learning. Most of the efforts have focused on the first two categories. In the workshop *Machine Learning for Creativity and Design* at NeurIPS 2018, for example, 35 papers were accepted, but only one paper addressed reinforcement learning [6] and the other one mentioned reinforcement learning in their future directions [7].

One reason that supervised and unsupervised learning attract more attention than reinforcement learning in the creative community is that the former two have a relatively straightforward workflow. For example, generative adversarial networks (GANs) [3] was originally proposed as a generative model for unsupervised learning, but it has been

adapted as a tool for creating artistic images. Basically, it can learn aesthetics from input images and generate novel images representing similar aesthetic characteristics. Once the models are constructed, the workflow for users is clear that they can focus on curating input images and post-processing generated images. The simplicity of the workflow also applies to other popular deep learning techniques used in artistic explorations, like pix2pix [8] or sketch-rnn [9].

For reinforcement learning, the workflow for art creation is unclear. As reinforcement learning is to solve sequential decision-making problems, the outputs are usually a sequence of actions, which is not feasible to be applied directly on tasks like image generation. In an RL problem, an agent starts by randomly exploring the environment and learns an adaptive strategy to solve the problem. What is the creative process of adapting reinforcement learning for artistic activities? What are the possible emergent behaviors that can be resulted from such a process? What are the artists' roles in the process? Those are questions that have yet been fully explored in previous research.

Reinforcement learning does share some similarities with generative art as essentially they are both driven by autonomous systems. Here I propose the term *RL-based generative art* to denote a novel form of generative art of which the use of RL agents is the key element in such form. This dissertation investigates a conceptual framework for RL-based generative art, which consist of the following components: a formal definition and its relation to generative art, a comparative study of the creative process between RL-based generative art and deep learning art, a discussion of artist roles and emergent behaviors in RL-based generative art.

In order to participate more effectively in reinforcement learning, creators also need a set of tools that can guide their activities. The existing machine tools for creators, like ML5 or Runway ML[1], primarily focus on supervised and unsupervised learning and

[1]Machine learning tools for creatives will be discussed in Chapter 2, Section 2.

provide a collection of trained models to help users to get started. Since each RL problem requires its own training environment to collect data, a trained RL model is not transferable to another RL problem. Even in the same RL problem, a small change in the environment, such as the number of actions, requires the model to be re-trained. Hence, it's pointless to provide pre-trained RL models to creators for artistic exploration as the models cannot be adapted to their own problems. On the other hand, current RL tools target on solving engineering-oriented problems or evaluating novel RL algorithms. This situation leads the dissertation to explore how to design an RL tool that is designated for artistic exploration. The tool should help developers focus on conceptually framing RL problems and rapidly prototyping RL environments.

1.3 Challenges

This section presents three main challenges when adapting reinforcement learning for generative art. Those challenges are not exclusively concerned by RL-based generative art but apply to reinforcement learning research in general. The following discussion focuses on their influences on the artistic exploration of reinforcement learning.

Training Efficiency

The recent successes of deep reinforcement learning rely on extremely expensive computational power and take days or even months to train. Dactyl[10], an AI system developed by OpenAI [2], used 384 machines that each machine contains 16 CPU cores to train a robotic hand to manipulate a cube. To achieve their desired results, the system requires about 100 years of experience to learn to rotate a cube in simulation. DeepMind [3]

[2] https://openai.com/
[3] https://deepmind.com/

trained RL polices to perform realistic locomotion behaviors[11] by using 64 machines for over 100 hours.

For artistic explorations, it's neither always necessary to achieve the optimal performance of a task nor it's realistic to have access to such expensive computational resources. One essential element in creative process is to rapidly prototype and test ideas. Experimenting with different combinations of parameters is a common approach. Current RL libraries require developers to have an in-depth understanding of advanced RL algorithms in which each algorithm usually consists of more than 20 hyper-parameters. A small discrepancy in hyper-parameters may result in failed training. For example, DDPG[12], a RL algorithm for continuous action spaces, uses target networks to stabilize updates that a target network only copies a small portion from its original network. This process is controlled by a hyper-parameter [4] τ, which is usually set to 0.001. In Ape-X DDPG, a distributed version of DDPG, τ is usually set to 1 to achieve the good performance as it uses another hyper-parameter *target network update freq* for stabilization.

Given the training inefficiency and the complexity of fine tuning RL algorithms, it's not feasible for creators to efficiently explore reinforcement learning if they simply experiment with hyper-parameters of RL algorithms. A more engaged approach needs to be discovered.

Environment Design

In supervised learning or unsupervised learning, once a model is trained, it can be used as a tool by taking curated data and generating aesthetic results. Such straightforward workflow can encourage creators to experiment with different models trained under the same technique. The workflow of reinforcement learning is inherently more complicated

[4] In deep learning, *parameters* refer to coefficients of a model. To differentiate, the term *hyper-parameters* is created to denote adjustable parameters in an algorithm.

because a training environment needs to be properly defined by developers to collect training data. That requires an in-depth understanding of RL concepts, like states, actions, and rewards. For example, to solve a CartPole problem[5], the agent needs to return the cart position, the cart velocity, the pole angle, and the pole velocity as states after the cart takes an action. This is considered as a fully observable environment. The problem is hard to be solved if one of the states is missing, unless a recurrent module is added to the neural network. In addition, debuggability is still an ongoing issue in deep learning that fixing a failed training heavily relies on the developer's experience. Thus, designing a RL environment from scratch is impractical for non-experts.

A collection of simple RL environments could be a possible solution, but the environments in the collection should be addable to a complex system for the purpose of experimenting RL environments. Currently, RL environments are usually not included in RL libraries that mostly focus implementing RL algorithms. Popular RL environments, like OpenAI Gym [13], are mainly designed for RL researchers to evaluate novel RL algorithms. Creators can benefit from a RL tool that focuses on simplicity and combines algorithms and environments into one framework.

Reward Function Design

Reinforcement learning uses a reward function to guide the agent to learn the desired behaviors without explicitly defining the rules. The benefit of learning is allowing the agent to be adaptive to uncertainties, but designing a reward function to precisely capture the developer's intention is challenging as misspecifying a reward function could result in absolutely unexpected behaviors.

Typically, if a reward is given only when the agent achieves the goal but no reward at

[5]A cart is either moving to left or right to balance a pole on top of it. The problem will be elaborated in Chapter 4, Section 3.

all other states, it's a sparse reward function, which is easy to define but could be very inefficient in training because unsuccessful attempts in sparse reward function do not contribute useful information for optimizations. In contrast to sparse reward function, shaped reward function could give feedback on every step taken by the agent, which is usually considered to make the policy easier to be learnt than sparse reward function.

However, a misspecified shaped reward function can lead to reward hacking to cause unintended behaviors. A simple example is to train an agent moving from left to right to reach a target in a 2D environment. Intuitively, a reward function could be designed as the closer the agent gets to the target, the higher positive reward it receives. If no other constraints, this reward function may result in the agent to roam around the target but never reach it. In that case, the agent finds out that reaching the target will stop receiving further positive rewards. Such unintended behaviors may result in some harmful consequences in real-world applications, but they are not always unfavorable conditions in the context of artistic explorations. That said, the challenge of designing a reward function for artistic exploration is not precisely capture the developer's intention, but to encourage the agent to develop more emergent behaviors. In the example in this paragraph, if a reward function is well defined, the agent could develop the optimal solution by moving straight to the target. However, this reward function is less interesting in comparison to a reward function that can guide the agent to develop different solutions every time it retrains. This area needs further studied.

1.4 Approaches

Since the concept of RL-based generative art hasn't been fully explored, this dissertation will initially provide a formal definition of RL-based generative art, clarify its creative process, and discuss the emergent behaviors in RL-based generative art. This

will lead to a discussion on several projects developed by the author in generative art, deep learning art, and reinforcement learning. The discussion will focus on their relation to RL-based generative art, conceptually and technically. With the new opportunities brought by RL-based generative art, the dissertation will develop a JavaScript library to improve the accessibility of reinforcement learning for creators. The library will be built on top of p5.js, a popular creative coding language. This allows developers to rapidly defining their RL environments and train RL policies without struggling with technical details of RL algorithms.

The library will be exercised with two pedagogical applications and two artistic applications. The pedagogical applications will use the library as a platform to introduce the concepts of RL and encourage participants to build their RL environments with the library. A quantitative survey will be conducted to evaluate the library. The artistic applications will explore different aspects of reinforcement learning. One will be a solo practice and the other one will be a collaborative project.

1.5 Structure of The Dissertation

This dissertation consists of six chapters. Chapter 1 *Introduction* describes the primary motivations, main challenges, problem statement, and approaches on bridging reinforcement learning and generative art. Chapter 2 *Background* starts with a general introduction of reinforcement learning, including Markov decision process and RL core concepts, followed by three short surveys on RL applications, RL libraries, and RL environments, respectively. The chapter then discusses a group of curated generative artworks from the 1950s to 2010, a group of curated deep learning generative artworks based on three deep learning techniques, and a selection of machine learning tools designed for creativity. Chapter 3 *Reinforcement Learning & Generative Art* introduces a conceptual

framework of how reinforcement learning can be adapted for generative art by proposing a term *RL-based generative art* to denote such art form and discussing the creative process and emergent behaviors in RL-based generative art. The chapter also discusses several author's practices on generative art and reinforcement learning. The practices explore various artistic experiments on generative art and technical improvements on reinforcement learning. Chapter 4 *RL5: A p5.js Library for Reinforcement Learning* introduces a JavaScript library that enables developers to train RL policies and prototype RL environments in web browsers. Specifically, the chapter describes three implemented RL algorithms, a collection of RL environments, core APIs, and a complete example of using the library to train and evaluate an RL agent. Chapter 5 *Practices* describes two pedagogical applications and two artistic practices of RL5. Lastly, Chapter 6 concludes with the results and future works of RL5.

Chapter 2

Background

This chapter is structured as follows. A general introduction of reinforcement learning (RL) is introduced in Sec 2.1, including Markov Decision Process and fundamental concepts of RL, followed by a brief survey of current RL applications in games and robotics, then a brief survey of current RL libraries and environments. In Sec 2.2, the scopes of generative art and deep learning art in this dissertation are defined, respectively. A group of curated generative artworks from the 1950s to the present is reviewed. A collection of deep-learning driven generative artworks is discussed based on three deep learning techniques: GANs, RNN, and CNN. Sec 2.3 reviews and discusses the current tools aiming to make deep learning accessible for creators and beginners.

2.1 Reinforcement Learning

2.1.1 Introduction

> *Reinforcement learning, one of the most active research areas in artificial intelligence, is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives when interacting with a*

complex, uncertain environment.

- *Reinforcement Learning: An Introduction* by R. Sutton & A. Barto

Reinforcement learning is a family of methods aimed at training an **agent** to collect rewards from an environment. At each time step t, the agent is given information about the **state** of its environment in the form of an **observation** s_t and then makes an **action** a_t. The agent's **policy** $\pi(s_t)$ is the logic that takes state observations and returns action selections. After each action, the environment will return a new state observation s_{t+1} and **reward** r_{t+1}. This cycle is illustrated in Fig. 2.1.

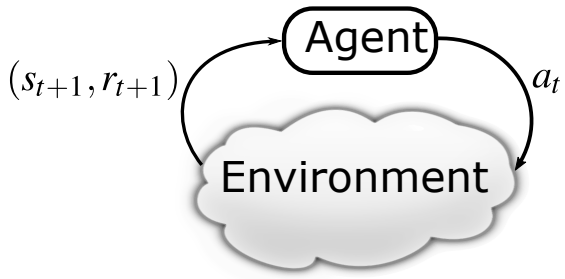


Figure 2.1: In reinforcement learning, an agent interacts with an environment. At each time step, the agent receives a state and reward signal from the environment. Based on this information, the agent selects its next action.

The agent's policy is parameterized by the tensor θ, giving it a more explicit notation of $\pi_\theta(s_t)$. For example, in a linear model, the policy would have the form:

$$\pi_\theta(s) = \theta_1 s_1 + \theta_2 s_2 + \cdots + \theta_m s_m = \theta^\top s, \tag{2.1}$$

where the time step t has been dropped for notation clarity and m is the number of features in the state space. In a similar manner, a neural network would be parameterized by a parameter tensor. The goal of the agent is to maximize collection of rewards from the environment. In a finite time-horizon, the goal is accomplished by finding parameters

$\theta^\star$ which provide this maximization:

$$\theta^\star = \arg\max_\theta \sum_{t=0}^{T-1} r(s_t, a_t), \tag{2.2}$$

where $T-1$ is the number of time steps experienced, and $r(s_t, a_t)$ is the environment's reward function. In many real-world cases the reward function is not given as a closed-form expression, but must be sampled by the agent's interactions with the environment. One of the strengths of reinforcement learning is its ability to learn using this experiential method.

Markov Decision Processes

Markov Decision Processes (MDP) are the environments which reinforcement learning was developed to solve. A major aspect of MDPs is that they have states in which an agent exists, and the outcomes of actions depend only on the current state, not on past states and actions; in this sense MDPs are **memoryless**. The memoryless property is captured in the environment's state-transition and reward function notation:

$$\begin{aligned} p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) &= p(s_{t+1}|s_t, a_t), \\ r(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) &= r(s_{t+1}|s_t, a_t). \end{aligned} \tag{2.3}$$

The state-transition and reward functions in Eq. 2.3 state that function outcomes depend only on the current state and action, and are independent of past states and actions. A second major aspect of MDPs is that the state-transition and/or reward functions may be **stochastic**, which means their return values are drawn from some underlying probability distributions. In standard RL settings, these distributions must be **stationary** which means the probabilities do not shift over time. Methods exist for using RL in nonstationary environments.

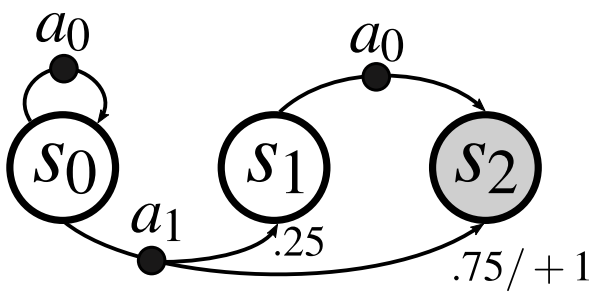


Figure 2.2: Example of Markov Decision Process. There are three states and two actions. Unless otherwise indicated, the state transition probability is 1 and reward is 0. Transition from s_0 to s_2 is the most interesting with $r(s_2|s_0, a_1) = 1$ and $p(s_2|s_0, a_1) = .75$.

Within an MDP, agents may observe their current state and make actions which attempt to affect the future state. The agent's objective is to maximize collection of rewards. An example of 3-state MDP[1] is given in Fig. 2.2. In this example, the initial state is s_0, and the agent has two action options: a_0 and a_1. If the agent chooses action a_0 it is guaranteed to stay in state s_0, denoted by $p(s_0|s_0, a_0) = 1$. If the agent chooses action a_1, there is a 25% probability that it will transition to s_1, denoted by $p(s_1|s_0, a_1) = .25$, and a 75% probability it will transition to s_2, denoted by $p(s_2|s_0, a_1) = .75$. The environment returns reward of 0 for all state transitions except for $s_0 \rightarrow s_2$, and in this case it returns $r(s_2|s_0, a_1) = 1$.

The agent's only goal is to maximize collection of rewards. In the context of Fig. 2.2 the agent should always select action a_1, as it is the only action that leads to a non-zero reward. While *we* can see that is the solution, an agent must learn it. There are two general approaches for learning in RL: **value iteration** methods and **policy gradient** methods. Value iteration is the classical approach to RL and includes the

[1]In the notation for this example, the subscripts denote "options", versus the usual meaning, which is time in this chapter.

Q-Learning algorithm and its descendents. Policy gradient methods directly optimize a policy through gradient ascent. Policy gradient methods perform well in many situations and they are relatively straightforward to implement.

Core Elements of Reinforcement Learning

A reinforcement learning system usually consists of the following core elements: an environment, one or multiple agents, a policy, states, actions, a reward function, a value function, and optionally, a model of the environment.

Environment & Agent: An RL environment hosts and interacts with one or more RL agents and is partially influenced by the agents. At every step of interaction, the agent observes a state of the environment and takes an action base on the state. After each step, the environment changes based on how the agent acts and also on its own. On the agent side, it perceives a reward from the environment, which is usually a numeric value indicating how the state is. The goal of the agent in RL is always to maximize its cumulative rewards. An environment could be either in simulation or reality, deterministic or stochastic, fully observable or partially observable, discrete or continuous, episode or sequential.

Reward Function: A reward function of an environment measures how good state-action pairs are. The RL agent receives a numerical reward every time step, and the goal of the agent is to maximize the total rewards, also called **return**, in the long run.

State & Observation: A **state** is a complete description of the environment at a particular step, and an observation is a partial description of a state. States or observations are always represented by a group of data, like pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game. If the agent can observe the complete state of the environment, then the environment is **fully observable**. Otherwise, the environment is **partially observable**.

Action Space: An action takes the agent from the current state to the next state. Action space is the set of all the valid actions and can be either **discrete** or **continuous**. Discrete action spaces indicate a finite number of actions are available, like Go, and continuous action spaces mean the actions are real-valued vectors, like the Cartesian-velocity controller of a robot. **Policy**: An RL policy maps from the received states of the environment to actions resulted from those states. Namely, it is a set of learned rules to guide an agent of what actions to take. It could be a look-up table, a simple function, or a sophisticated neural network. It could be deterministic or stochastic that the latter allows the agents to explore the environment without always taking the same action.

Value Function: In comparison to reward function, which evaluates the RL agent's behavior in an immediate sense, a value function evaluates what is good in the long term. The value of a specific state reflects the expected total rewards from that state to the end of the episode. Value functions are used in almost every RL algorithm.

Model of the Environment: A model of an RL environment calculates the dynamics of the environment. Given a state and action, the model will predict the next state, denoted by $p(s_{t+1}|s_t, a_t)$. Depending on if a model is used during the training, reinforcement learning can be categorized as model-free RL and model-based RL.

2.1.2 RL Applications

Reinforcement learning has been primarily applied in many fields of which games and robotics are most relevant to the creative community. Hence, a curated collection of the popular RL applications in those two fields are listed below.

Games

Games are great but challenging domains for testing RL algorithms. On the one hand, games are ideal training environments as the rules are well defined and the moves involve sequential decision-making progress. On the other hand, without any enhancements, the basic reinforcement learning algorithms are insufficient to solve games due to rich search space.

AlphaGo [5], developed by DeepMind to play the board game Go, is considered as one of the most well-known applications of RL. The game of Go is known as the most challenging classical games for artificial intelligence because of its numerous search space. Previously, the strongest Go computer programs could only beat amateur human Go players. AlphaGo is the first computer program to defeat a Go world champion. It uses one deep neural network to evaluate board positions and another deep neural network to select moves. The networks are trained by a combination of supervised learning from human expert games and reinforcement learning from games of self-play. Later, DeepMind released a newer version called AlphaGo Zero [14], which did not use any prior knowledge of the game and took three days of self-play train to beat AlphaGo. The significance of this version is that it frees itself from the constraints of human knowledge, and it only used a single machine with 4 TPUs.

Before AlphGo, DeepMind presented DQN [15]: "the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning." The same architecture of the model and the same algorithm were used to train seven Atari 2600 games. The results show that "it outperforms all previous approaches on six of the games and surpasses a human expert on three of them." The states are raw pixels from the screen, and game scores and the actions are applied directly to the game agents. DQN is an off-policy model-free RL algorithm that the

model periodically samples transitions from a replay buffer for optimization.

Besides board games and classic video games, reinforcement learning is also studied in modern multiplayer online battle arena (MOBA) video games, like Dota or Warcraft. OpenAI Five [16] is a five-neural-network RL system designed to beat top human professional teams at Dota 2. Its early version uses another off-policy model-free RL algorithm called Proximal Policy Optimization (PPO) [17] to train on 256 GPUs and 128,000 CPU cores. On April 13th, 2019, OpenAI Five "became the first AI system to defeat the world champions at an esports game." In this version, they developed "a distributed training system and tools for continual training which allowed us to train OpenAI Five for 10 months", which demonstrates that "self-play reinforcement learning can achieve superhuman performance on a difficult task."

DeepMind chose another challenging Real-Time Strategy (RTS) game StarCraft II as the testbed for their novel reinforcement learning system named AlphaStar [18]. Unlike OpenAI Five that crafts information available to a human player in a set of data arrays, AlphaStar uses the raw game interface as state space. It was ranked above 99.8% of active players on Battle.net, the official game server of StarCraft II. To train a satisfying version, it took about 44 days running concurrently 16,000 matches and 16 actor tasks that each uses a TPU device with eight TPU cores.

Robotics

Robots have been heavily used in manufacturing and other industries; however, as they rely on pre-defined trajectories, they require precise calibration and fail to adapt to uncertainties. Adaptability to imprecision, varying conditions, and less structured environments is key to the future of automation. As most robotic problems can be framed as sequential decision-making problems, robotics can be seen as an important and challenging application platform for reinforcement learning to test in the real world.

Recently, reinforcement Learning has led to a series of successes in solving robotic tasks, like assembly, control, or navigation.

Reinforcement learning has been studied actively in the area of high precision assembly as it can reduce human involvement and increase the robustness to uncertainties. Inout el al. [19] used a Q-learning based method with LSTM [20] for Q-function approximation to solve low-tolerance peg-in-hole tasks. Luo el at. [21] extended a model-based approach MDGPS [22] with haptic feedback for learning the insertion of a peg into a deformable hole. Fan el at. [23] combined DDPG [24] and GPS [25] to take advantage of both model-free and model-based RL [26] to solve high-precision Lego insertion tasks. Luo el at. [27] combines iLQG [28] with force/torque information by incorporating an operational space controller to solve a group of high-precision assembly tasks.

Training efficiency is a significant challenge of training real robots with reinforcement learning. Experience replay [29] has been used to improve training efficiency, particularly for model-free RL algorithms, as it is less sample efficient than model-based RL. The technique has become popular after it was incorporated in the DQN [30] agent playing Atari games. Prioritized experience replay [31] is a further improvement to prioritize transitions so agents can learn from the most "relevant" experiences. Hindsight experience replay [32] stored every transition in the replay buffer not only with the original goal but also with a subset of other goals to acquire a more generalized policy. DDPG from Demonstrations [33] modified DDPG to permanently store a set of human demonstrations in the replay buffer to solve a group of insertion tasks.

2.1.3 RL Libraries and Environments

Practically, to perform an RL task, an RL environment needs to be defined and an RL algorithm needs to be selected to match the action and state spaces of the RL envi-

ronment. For example, a pendulum environment has to be trained by an RL algorithm supporting continuous action spaces. An RL library usually refers to a set of implementations of RL algorithms, and RL environments are usually developed and distributed separately from RL libraries.

A Survey of RL Libraries

A lot of deep reinforcement learning libraries have been developed, but it is hard to determine the best one. This section gives a preliminary analysis based on the following criteria: implemented RL algorithms, documentation, flexibility with different RL environments, and accessibility of the codes. Fig 2.3 presents a non-exhaustive taxonomy of RL algorithms, adapted from *OpenAI Spinning Up*[2]. Most of the RL libraries implemented model-free RL algorithms because the transition model of training environments could be agnostic, which improves the reusability of the algorithms.

OpenAI Baseline [34] implements a set of high-quality RL algorithms with TensorFlow, including A2C, ACER, ACKER, DDPG, DQN, GAIL, HER, PPO, and TRPO. However, the library neither comes with documentation nor the codes are commented. It is naturally compatible with OpenAI Gym environments but remains unclear of how to work with external environments. Since each algorithm was implemented in different styles, the codes were also challenging to be modified.

Tensorforce [35] is a deep reinforcement learning framework focusing on modular component-based design. Similar to OpenAI Baseline, it implements a majority of state of the art RL algorithms, including A2C & A3C, DQN and its improvements, Policy Gradient, PPO, and TRPO. The library misses DDPG, which is a popular algorithm for continuous action space. The library makes its algorithms agnostic to the type and structure of state and action, as well as the training environment, which is friendly for

[2] `https://spinningup.openai.com/en/latest/index.html`

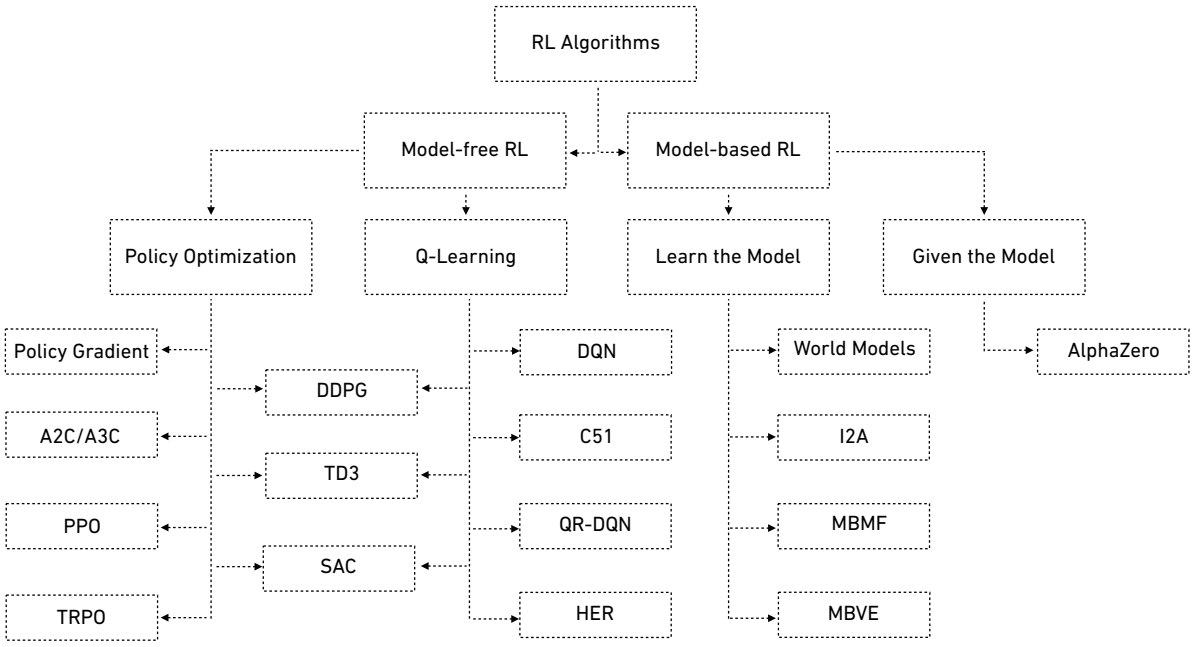


Figure 2.3: A non-exhaustive taxonomy of RL algorithms, adapted from *OpenAI Spinning Up*.

external environments. The modular design improves the accessibility to modify the codes of which more comments are needed.

KerasRL [36] was designed to integrate with the deep learning library Keras seamlessly. It implemented CEM, Continuous DQN, DQN, DDPG, and Deep SARSA, but it misses two important methods, Actor Critic and Proximal Policy Optimization. The codes are well-documented but are explicitly designed to work with OpenAI Gym. Therefore, modifications to the codes are required in order to use external RL environments. On the other hand, due to the separation between agents, policies, and memory, the modifications are relatively straightforward.

Stable Baselines [37] was developed on top of OpenAI Baseline as an improvement by providing a unified structure for all the algorithms. Besides the algorithms implemented by OpenAI Baseline, Stable Baseline includes SAC and Double DQN. It also creates a collection of pre-trained reinforcement learning agents, called RL Baseline Zoo. The library

provides a tutorial of how to train on self-defined environments, but the environments have to be wrapped as Gym-type environments.

Besides the aforementioned libraries, the most recent RL libraries are focusing on improving either scalability, like rllib [38] (a framework for distributed reinforcement learning), or compatibility, like rlpyt [39] (offering a shared and optimized infrastructure to build all three RL algorithm families). However, very few RL libraries were designed specifically for the creative community.

A Survey of RL Environments

In an RL system, an RL environment is as equally important as an RL algorithm as an environment needs to define state spaces, action spaces, reward function, and dynamic transitions. A small discrepancy in the environment design could lead to entirely different training results. Current available RL environments are primarily designed to test RL algorithms.

OpenAI Gym [13] is a diverse collection of RL environments, including toy texts, classic control, algorithmic, Atari, board games, and 2D and 3D robots. It was created for developing and comparing reinforcement learning algorithms. Since the environments have been released, they have become benchmarks for RL researchers to test and compare RL algorithms in RL literature. Below is a detailed introduction of the RL environments OpenAI Gym implemented:

- *Classic control & toy text* are small-scale but complete tasks from RL literature, like CartPole, Pendulum, Acrobot.
- *Algorithmic* includes simple computational tasks, like adding multi-digit numbers or reversing sequences.

- *Atari* plays classic Atari games, integrated from Arcade Learning Environment, which will be introduced next.

- *2D & 3D robots* simulates robots in 2D or 3D environment, using the MuJoCo [40] physics engine, "which was designed for fast and accurate robot simulation".

Arcade Learning Environment (ALE) [41] provides an interface to hundreds of Atari 2600 games for evaluating domain-independent agents of reinforcement learning or imitation learning. Specifically, "ALE provides a game-handling layer, which transforms each game into a standard reinforcement learning problem by identifying the accumulated score and whether the game has ended. By default, each observation consists of a single game screen (frame): a 2D array of 7-bit pixels, 160 pixels wide by 210 pixels high. The action space consists of the 18 discrete actions defined by the joystick controller."

Taking one step further from ALE, Retro Learning Environment (RLE) [42] introduces a framework to enable training on the Super Nintendo Entertainment System (SNES), Sega Genesis, and several other gaming consoles. Developed and released by Nintendo in 1990, SNES is a video game console consisting of 783 games, which include some iconic ones, like *Super Mario World*. SNES increases the level of complexity and versatility of the video games for RL algorithms. Besides the defaults, more games and consoles can be added to the environment with the same interface as ALE, and the environment is compatible with Python and Torch.

AI Safety Gridworlds [43] is a suite of RL environments focusing on safety perspective of intelligent agents, like safe interruptibility, avoiding side effects, absent supervisor, or reward hacking. They have designed different environments for each safety perspective, but all the environments "use a grid of size at most 10x10. Each cell in the grid can be empty or contain a wall or other objects. These objects are specific to each environment and are explained in the corresponding section. The agent is located in one cell on

the grid. In every step, the agent takes one of the actions from the action set $A = \{left, right, up, down\}$. Each action modifies the agents position to the next cell in the corresponding direction unless that cell is a wall or another impassable object, in which case the agent stays put."

AI Habitat [44] and AirSim [45] both focus on providing photo-realistic and high-performance 3D simulator as an attempt to fill the gap between simulation and reality. AI Habitat has built-in support for Matterport3D [46] and Gibson [47]. It can render a scene from Matterport3D with several thousand frames per second on a single thread and over 10,000 frames per second on multiple threads, both running on a single GPU. AirSim, on the other hand, builds on Unreal Engine to focus on simulating drones and cars. "It is open-source, cross-platform, and supports hardware-in-loop with popular flight controllers such as PX4 for physically and visually realistic simulations."

Unity ML-Agents Toolkit [48] provides a collection of environments implemented in Unity that "contains examples of single and multi-agent scenarios, with agents using either vector or visual observations, taking either discrete or continuous actions, and receiving either dense or sparse rewards." Developers can use those environments to design their environments in Unity. Additionally, it also implemented two RL algorithms, PPO and SAC. Conceptually, Unity ML-Agents is mostly close to the goal of this dissertation. However, it is primarily for game designers and the learning curve of using Unity and the toolkit is also steep.

2.2 Deep Learning & Generative Arts

2.2.1 Generative Art

Definition of Generative Art

Generative art is a broad topic that could mean various art forms. In the scope of this dissertation, generative art refers to the definition proposed by Philip Galanter:

> "Generative art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art." [49]

In Galanter's definition, the core element of a generative artwork is an autonomous system that is not directly controlled by the artist but is derived from a set of rules or a logic structure. The system ranges from order to disorder, like symmetry and tilling, fractals, L-system, genetic systems, stochastic L-system, stochastic fractals, or randomization. Thus, generative art can be categorized as "highly ordered generative art, highly disordered generative art, and complex generative art". The following section mostly introduces complex generative art in which the system combines order and disorder.

A Brief History of Generative Art

This section lists a group of curated generative artworks from the 1950s to the present to show the evolution of generative art.

Pithoprakta(1955-56) is a generative composition for string orchestra that each instrument is treated as a gas molecule follows the Maxwell-Boltzmann distribution law [50]. An analogy was drawn between a string instrument through its pitch range and the movement of a gas molecule through space. The artwork was created by Iannis Xenakis, who

is recognized as one of the early pioneers of using mathematical models for music. The whole system was manually calculated by Xenakis himself rather than by a computer. On the highest level, although each component of the work was resulted from the generative system, the structure of the work was organized under the artist's intention.

23-Ecke(Polygons of 23 Vertices, 1965) is a computer graphic created by randomly generating polygons that each polygon consists of 23 vertices, alternatively "chosen in horizontal or vertical directions within the grid cell. The first and the last vertex are connected in whichever direction is determined by their positions."[51]. The process led to the discovery of new images, and the graphic displayed the effects of change. Georg Nees, the creator of the piece, is considered as one of the founders of computer art. The project was based on the programming language ALGOL and the Siemens computer system 4004.

Random Ploygon 13/9/65 Nr.7.(1965) was created in the same year as ***23-Ecke*** by Frieder Nake. The algorithm randomly chooses a direction and generate the length along that direction. "Only a discrete set of directions was permitted: horizontally left or right, diagonally in 60 or 120 degrees, both up and down, or more or less vertically within a small range of directions around the vertical. Lengths were also to be taken from a short distance range, a middle, and a long distance range." [51] Although the two artworks are visually different, both of them explored polygons by randomly determining the positions of the vertices within a certain set of rules.

Calculated Movements(1985) is a 6-minute short film [52] to show a sequence of movements of 3D geometric shapes. Each element in the film follows a simple rule in motion, and the group of motions together creates a complex visual composition changing over time. The film was created by Larry Cuba using a programming language called GRASS, explicitly made for transnational and transformational animation.

Standing Figure with Decorated Background(1995) is a painting created by

a robot painter AARON, which is considered as the first robot in human history to paint original art and the oldest continuously developed program in computing history. AARON was developed by Harold Cohen starting from the 1970s. The breakthrough idea of AARON was that a system makes its own decision based on its knowledge base, "declarative knowledge and procedural knowledge" [1]. The system has to build its internal representation of the developing drawing in order to make the next decision.

Pockets Full of Memories(2000-2007) is an interactive installation that uses the Kohonen self-organizing map algorithm to organize a database of objects by positioning the objects of similar value near each other in a two-dimensional map. The objects were collected from visitors by digitally scanning from a kiosk-like scanning station and describing them from a designed questionnaire. The installation was developed by George Legrady and Timo Honkela, and was premiered on the main floor of the Centre Pompidou in 2001 and attracted approximately $20,000$ visitors who contributed over $3,300$ objects in their possession. Later, the project was featured in Ars Electronica, Linz (2003), Aura, C3 Center for Culture & Communication, Budapest (2003), Museum of Contemporary Art Kiasma, Helsinki (2004), Cornerhouse Gallery, Manchester (2005), Frankfurt Museum of Communication (2006), and Museum of Contemporary Art, Taipei (2007).

2.2.2 Deep-learning Generative Arts

Definition of Deep-learning Generative Art

Based on the definition in this dissertation that generative art is driven by a system consisting of a set of rules, deep-learning generative art indicates that the rules are learned by deep-learning techniques, but not directly defined by artists. Therefore, the system behind deep-learning generative art is usually a deep generative model [53]. One of the popular generative models is generative adversarial networks (GANs), introduced

by Goodfellow in 2014 [3]. In contrast to conventional generative art of which the aesthetics are realized by specific rules defined by artists, GANs learns aesthetics from the given dataset and generates new data, such as images or sounds, that follow the learned aesthetics.

As summarised by Elgammal, the creative process in the procedure of creating deep-learning generative art is "primarily done by the artist in the pre- and post-curatorial actions, as well as in tweaking the algorithm." [54] Namely, artists practice deep-learning generative art by preparing training data, adjusting hyperparameters of deep learning algorithms, and selecting generated data to form a presentation. However, this pipeline is not suited for agent-based generative art where the art is generated by the behaviors of the agent(s) in the system. On the other hand, reinforcement learning can be seen as an agent-based generative model, which has the potential to contribute to agent-based generative art. The relationship between agent-based generative art and reinforcement learning will be discussed in Chapter 3.

The next section lists a collection of recent deep-learning driven artworks from three different generative models in deep learning, which are GANs, Recurrent Neural Network (RNN) [4], and Convolutional Neural Network (CNN) [55].

GANs

GANs was initially invented as a generative model for unsupervised learning that the system consists of a *generative network* to generate increasingly convincing output and a *discriminate network* to distinguish real data from the artificially generated data. The technique can generate new data with the same statistics as the given training dataset. A famous example is NVIDIA's hyper-realistic face generator [56].

In art creation, GANs has been widely adapted by artists to examine creativity and culture. ***Portraits of Imaginary People***(2017), created by Mike Tyka, explores high-

resolution GANs-generated faces that focus on fine details of texture. Vanilla GANs usually produces images in the size of 256 x 256 pixels, while Tyka aims to create 4k printable images. To realize the goal, Tyka used a three-stage GANs architecture with roughly 20,000 high-resolution portraits from Flickr as his training dataset. ***Memories of Passersby I***(2018), created by Mario Klingemann, is an autonomous system that uses multiple GANs to generate a sequence of never-ending and never-repeating artistic portraits of non-existing people. The system was trained based on thousands of portraits from the 17th to 19th centuries, and its aesthetic preferences, guided by the artist, were influenced by surrealist figures. The transformation between two images reveals the "thinking and struggling process" of the machine in real-time.

RNN

RNN is a kind of deep neural network where an output is not only determined by its current input but also affected by prior input(s). Namely, RNN can process temporal information, so it has been largely applied to tasks like speech recognition or handwriting recognition that only one input is usually ambiguous. The concept of RNN was originally developed from Rumelhart and his colleagues' work in 1988 [4]. Since 2014, Long Short-Term Memory (LSTM) [20], an advanced architecture of RNN, has broken many records in speech recognition [57], machine translation [58], and language modeling [59].

Artists usually take advantage of RNN's memories to embed temporal information into generative art. ***Drawing Operations Unit: Generation 2 (Memory)***(2018) is a robotic art created by Sougwen Chung. The project was a collaboration between the artist and her robot that they drew simultaneously on a canvas, and the robot mimicked the artist's gestures from her style and past movements. The outcome was a painting presenting a flower-like structure composed of curvy blue line. ***I Touch You and You Touch Me***(2016-2017) is another example of practicing RNN in robotic art. Created by

Jeffrey Thompson, this project explores human-machine daily interactions by feeding a robotic arm all the interactions between the artist and his cell phone in an entire month for the robot to learn the artist's gestures, like swipes, taps, or typing. In both of the works, the artists explored how to work with the advancements in AI in their artistic endeavors, rather than how AI are trying to remove the need for human intervention in art creation.

CNN

CNN is similar to an ordinary neural network but makes an explicit assumption that the inputs are images. CNN started to get public's attention in 2012 when AlexNet achieved a top-5 error of 15.3% in the ImageNet 2012 Challenge. [55] Since then, the technique has been widely applied to image recognition, video analysis, and natural language processing.

One research direction of CNN is to understand the decision-making process of CNN models by visualizing each layer in the model. This scientific research also results in many artworks. ***DeepDream***(2015), created by Google engineer Alexander Mordvintsev, was a byproduct from a research to visually understand the emergent structure of deep neural networks [60]. After a CNN model is trained, it can be run in reverse to adjust the original image for yielding a higher confidence score. The adjusted images often show a hallucinogenic appearance. ***DeepDream*** is always considered as one of the first artistic applications of deep learning.

As it highly relates to vision, another research direction in CNN is exploring algorithmic understandings of how humans create and perceive images. *A Neural Algorithm of Artistic Style* [61] directly targets artistic imagery by creating a system to separate content and style of arbitrary images to achieve artists style transfer. This technique has also been widely adapted by artists. One example is *Cubist Mirror*(2015) created

by Gene Kogan. It is an interactive installation feeding live stream from a webcam to a trained neural network, which outputs real-time videos in the Cubist style.

2.3 Creative Tools to Deep Learning

This section discusses popular creative coding tools specifically designed for artists to explore machine learning. Tools, like TensorFlow or PyTorch, can be used by artists to practice machine learning, but they are not in the scope of this discussion because their general purposes are not for artists. This goal of this section is not to provide an exhaustive survey of all the creative tools to deep learning, but to highlight the most foundational design choices in deep learning tools for artists. The tools discussed in this section reflect the progressions of ideas from the recent history of the field and also expose some of the trade-offs in designing deep learning tools for artists.

2.3.1 p5.js & ml5.js

p5.js is a JavaScript library created by Lauren McCarthy and maintained by the Processing Foundation[3] and NYU ITP[4], aiming to make coding accessible for creatives and beginners [62]. Not originally designed for machine learning, p5.js is a powerful tool for visual art and has been chosen by many artists as their entry point for creative coding. After it was released in 2014, "it has been used and integrated into curricular around the world".

One advantage of using p5.js, or JavaScript in general, is its ubiquitous support by all the modern web browsers so that p5.js programs can be run in any web browser. Feasibility is another feature of p5.js to help developers rapidly create prototypes. Vanilla

[3] `https://processingfoundation.org/`
[4] `http://itp.nyu.edu/itp/`

p5.js supports basis data structures, drawing functionalities in 2D and 3D, and simple interactions and animations. In *Nature of Code* [63] Chapter 10, Daniel Shiffman demonstrated how to build a basic neural network using native p5.js and several visualizations of how the neural network function. As the growing of p5.js community, it also provides a large number of libraries to support sound, physical simulation, external hardware, and so forth.

ml5.js also a JavaScript library specifically designed to make machine learning approachable for non-experts [64]. Initially developed and maintained by NYU's Interactive Telecommunications/Interactive Media Arts program, ml5.js was built on top of tensorflow.js and provides a collection of pre-trained models on images, sounds, and texts. Users can experience real-time human pose estimation, real-time object detection, image-to-image translation, or auto draw. Since the models are pre-trained, it helps users to skip the algorithmic part of machine learning and focus on the creative part.

Since p5.js and ml5.js are both browser-based and p5.js heavily influences ml5.js' APIs, the combination of the two libraries create an accessible web platform to apply machine learning techniques in visual art. However, it currently focuses on supervised and unsupervised learning but does not include any contents of reinforcement learning.

2.3.2 Magenta.js

Magenta.js is a JavaScript library developed by Google AI that intends to make machine learning more accessible by "abstracting away technical details, making it easier than ever for app developers to create new interfaces to generative models." [65] The long-term goal of Magenta.js is to provide a suite of compositional tools, and their current packages focus on music, sketch, and image of which the latter two are at their early development.

For Magenta.js/music package, it contains a JavaScript API for experimenting with note-based music generative models. The API supplies five model classes, MusicRNN, MusicVAE, DrumsRNN, PerformanceRNN, and ImprovRNN, and developers can choose either to use a pre-trained model or to train their models. Some implementations include converting raw audio to MIDI, or mapping 8-button input to a full 88-key piano in real-time. The library has a parallel Python version that also primarily focuses on generative models in music. The Magenta's documentation mentions briefly about developing reinforcement learning algorithms for generating songs and images, but no RL contents were found in their source codes and demos.

2.3.3 Runway ML

Runway ML is a standalone application also aiming to provide easy access to machine learning for creators. Unlike ml5.js or Magenta.js, Runway ML tries to be a platform for people without any coding experience to use machine learning in intuitive ways. The main feature of the application is providing a drag-and-drop interface to explore machine learning models from their curated model directory, which consists of motion capture, object detection, masking, style transfer, text synthesis, and audio generation. The interface also supports image, video, and text as inputs and outputs. Besides the standalone version, it can be integrated with Unity and Adobe Photoshop.

The efforts of entirely removing coding from the workflow further reduce the barrier between creators and machine learning, but it does not help to understand the mechanism of machine learning models. Currently, the application does not support training from users' data, and it does not have any plan to integrate reinforcement learning.

2.4 Summary

This chapter provides a high-level introduction to reinforcement learning, including the core concepts of reinforcement learning, the timely popular RL applications in games and robotics, and the state-of-art RL libraries and environments. It also examines system-based generative art and deep-learning generative art by definitions and examples. Lastly, the chapter discusses several creative tools to make deep learning more accessible to artists. From the reviews, most of the efforts focus on applying supervised learning and unsupervised learning to generative art but barely touch reinforcement learning. As a generative model, however, an RL policy has a great potential for generative art. In the next chapter, the relationship between reinforcement learning and generative art will be discussed and a conceptual framework of adapting reinforcement learning for generative art will be introduced.

Chapter 3

Reinforcement Learning & Generative Art

In this chapter, an attempt is made to offer a conceptual framework of adapting reinforcement learning for generative art. First, the author proposes the term *RL-based generative art* to denote a novel form of generative art of which the use of RL agents is the key element. A formal definition of RL-based generative art is also provided. Classic autonomous agents in generative art follow a set of rules explicitly defined by artists, while RL agents learn their behaviors under the guidance of reward functions. This leads to a discussion of the creative process, the role of artists, and emergent behaviors in RL-based generative art. Finally, several authors practices on generative art, deep-learning art, reinforcement learning, and their relations to RL-based generative art are discussed. The practices have been critical parts for adapting reinforcement learning for generative art in both conceptual and technical aspects.

3.1 Definition of RL-based Generative Art

According to Galanter's definition, the use of autonomous systems is a key element in generative art that artists partially or entirely define the rules of the systems. When a system contains autonomous agents, it derives a subclass of generative art that the artworks are generated by a sequence from behaviors of the autonomous agents in the system. An autonomous agent is considered as an entity "that makes its own choices about how to act in its environment without any influence from a leader or global plan" [63]. For example, artworks resulting from random walk [66], game of life [67], or diffusion-limited aggregation [68] should be categorised in this subclass. Deep-learning artworks like *DeepDream* or *Portraits of Imaginary People* should not be considered in this subclass because the outcomes were not from a sequence of behaviors and the concept of autonomous agents were not clear in those scenarios. Here is an attempt to name this subclass as *agent-based generative art* and the definition is:

> Agent-based generative art is a subclass of generative art where the outcomes are generated by a sequence of behaviors from one or multiple autonomous agents in an environment, and the behaviors are controlled by a set of rules partially or entirely defined by artists.

In comparison to classic autonomous agents in generative art that follow a set of explicit rules, reinforcement learning agents can be seen as a kind of autonomous agents that learn the rules by interacting with their environments. Here the author proposes the term *RL-based generative art* to denote a novel form of generative art, which can be defined as:

> RL-based generative art is a subclass of agent-based generative art that the outcomes are generated by a sequence of behaviors from one or multiple RL-

agents in an environment, and the behaviors are learned under the control of reward functions defined by artists. An RL agent does not assume knowledge of an exact mathematical model but learns its behaviors by taking actions in an environment to maximize some notion of cumulative reward.

Fig 3.1 illustrates the relationship between generative art, agent-based generative art, and RL-based generative art. The main difference between RL-based generative art and other agent-based generative art is whether the rules are directly defined by the artist or learned from a reward function defined by the artist.

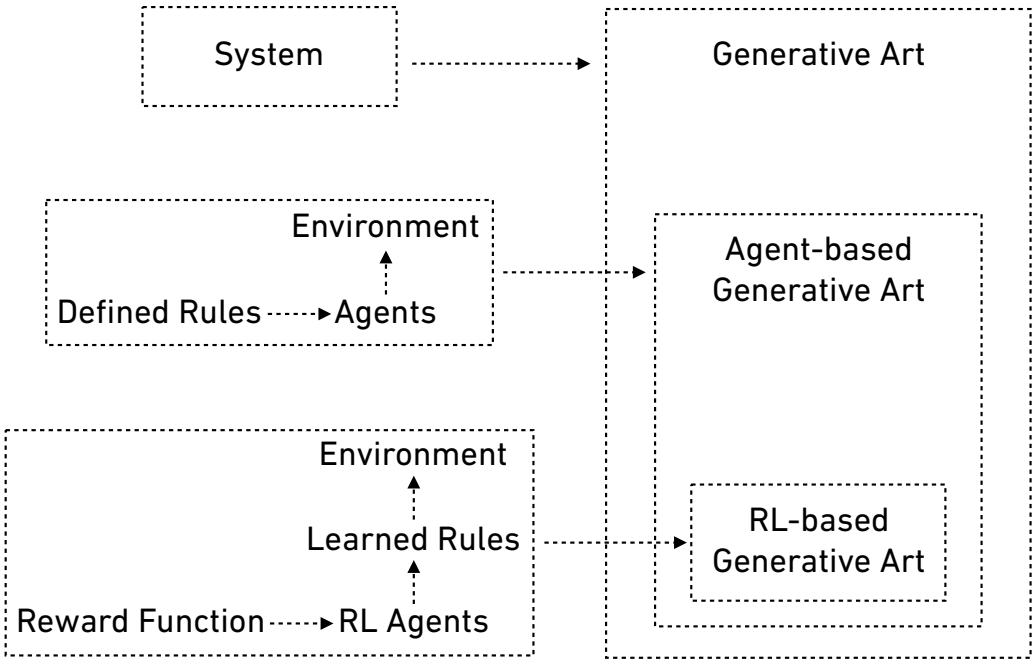


Figure 3.1: The relationship between generative art, agent-based generative art, and RL-based generative art.

3.2 Creative Process of RL-based Generative Art

3.2.1 Creative Process & Roles of Artists

Before introducing the creative process of RL-based generative art, it is worth to review the creative process of the current deep-learning art proposed by Ahmed Elgammal [54], as depicted in Fig 3.2. In the process, the artist usually curates a dataset to feed to a trained generative model, which outputs generated data. The artist can choose to fine-tune some hyper-parameters of the model for more desirable results, and present the final artwork by curating the generated outcomes.

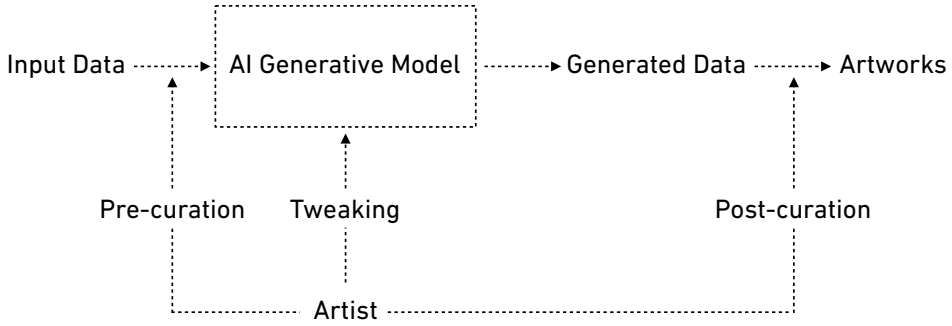


Figure 3.2: The *AI-Art Creative Process* proposed by Ahmed Elgammal, presenting the standard workflow of current deep-learning art.

In this pipeline, the creative process is mostly in the pre-curation and post-curation phases. Many great artworks have been created with the pipeline, but artists have minimal control over the generative models. This process is arguably missing the artist's intention and is more generative than creative. In some GAN-generated portraits, for example, the deformations in the faces carry a certain aesthetics that remind people of some Francis Bacon's works. However, Bacon intended to deform the faces in his works, but the deformations in those GAN-generated portraits were not from the artist's original intention. The outcomes were actually caused by the generative model's unsuccessful imitations of human faces.

In RL-based generative art, as the training data are collected during the interactions between the RL-agent(s) and the environment hosting the agent(s), artists need to design a training environment rather than directly choose a set of training data. The engagement of the concepts of reinforcement learning and its technical details make RL-based generative art less accessible than other deep-learning generative art, but allow artists to gain more controls during the creative process. Fig 3.3 shows the creative process of RL-based generative art in which an essential role of artists is to design an environment consisting of visuals to present aesthetics, a reward function to guide the behaviors of the agent(s) in the environment, and transition dynamics to determine the interactive mechanism between the agent(s) and the environment. Unlike the deep-learning generative art discussed earlier, the process of designing an RL environment is inherently creative.

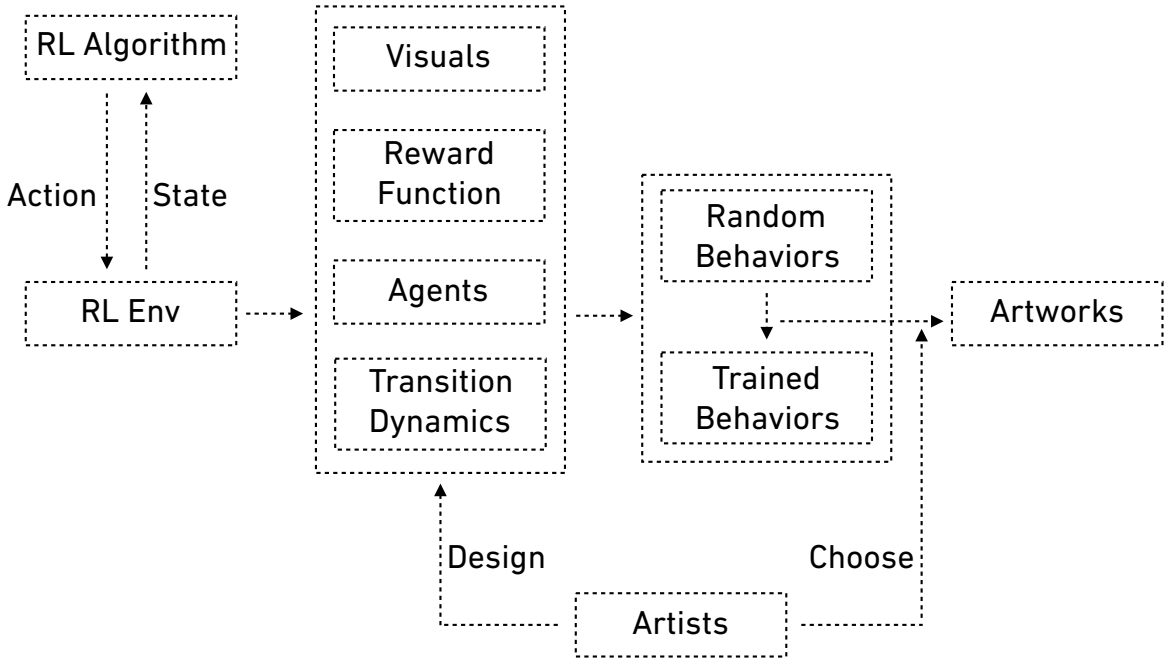


Figure 3.3: The creative process of RL-based generative art. In the process, the artist needs to design an environment, including visuals, a reward function, agents, transition dynamics, and state and action spaces. In addition, the artist has to decide how to present the environment from random behaviors to trained behaviors.

Besides designing an RL environment, another role of artists is to decide on how to present the environment as the final artwork. Illustrated in Fig 3.4, the evolution of an RL system usually consists of three phases: from a random system before training starts, to a chaotic system in the training when the agent explores the environment, to finally become an adaptive system when training finishes. Both of the behaviors in a random system and a chaotic system are unpredictable, and they are non-linear. The difference between the two systems is that the latter is driven by a deterministic machine, which is an evolving RL policy in this case. The difference between an order system and an adaptive system is that the latter can handle uncertainties in a stochastic environment. For example, in a peg-in-hole task, a robotic arm tries to insert a peg into a hole. If the position of the hole never changes and the robotic arm always starts at the same pose, it is hard to tell if it is an order system or an adaptive system. However, if the position of the hole shifts at every attempt, an adaptive system could still accomplish the task, while an order system might fail at most attempts.

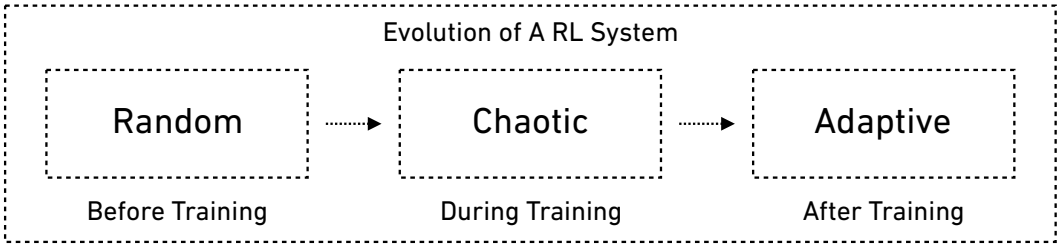


Figure 3.4: Three phases in an RL system: a random system in the before-training phase, a chaotic system in the during-training phase, and an adaptive system in the after-training phase.

Since each phase of an RL system and their combinations can be adapted for generative art, RL-based generative art creates a rich space for artists to explore new opportunities. For example, the training process from random explorations to trained behaviors has artistic potentials to present the evolution of an AI system. An abstract painting

could be created by the footprints of multiple trained RL agents in a stochastic environment. More sophisticated structures or frameworks will be coming in the future as more artists explore RL-based generative art.

3.2.2 Emergent Behaviors

One intriguing part of generative art is to produce emergent phenomena from the system. This section discusses emergent behaviors that could be possibly triggered by three core elements in RL-based generative art, *random exploration*, *state and action spaces*, and *reward function*. To facilitate the discussion, an RL environment called *Avoidance* is created as the example. The description is below.

Goal: to train an agent moves to the right end of a 2D canvas without hitting the rectangle in the middle, as shown in Fig 3.5.

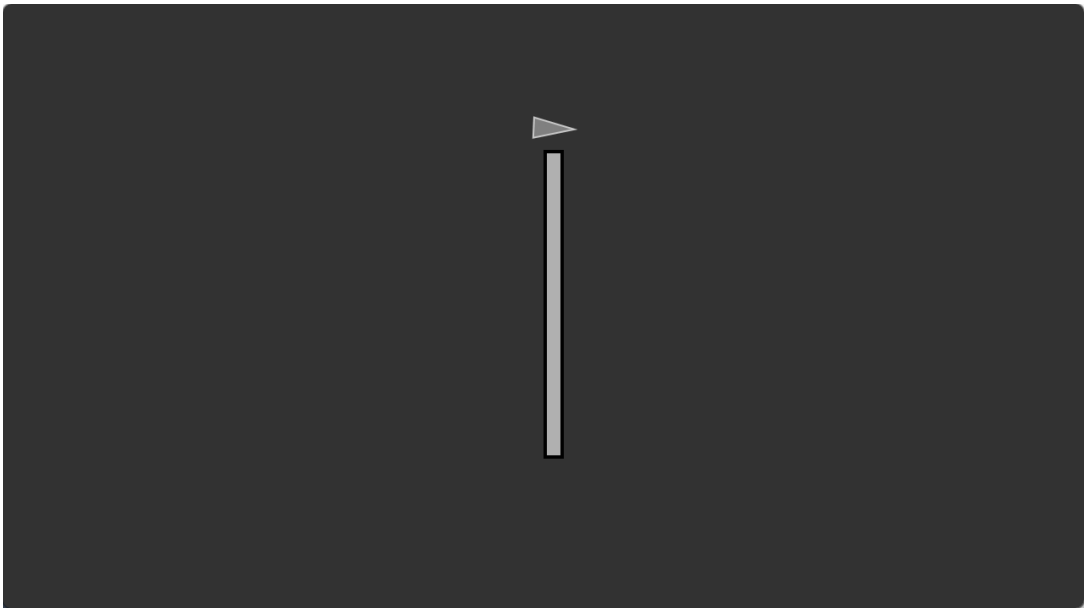


Figure 3.5: An RL environment called *Avoidance* in which an RL agent tries to move to the right end from the left end without touching the rectangle in the middle.

States	Min	Max
Normalized X Coordinate of the Agent	0.0	1.0
Normalized Y Coordinate of the Agent	0.0	1.0

Actions	Values
Push the agent to the top	0
Push the agent to the bottom	1

Terminal States
The agent hits the obstacle
The agent reaches one of the boundaries
The episode reaches a maximum number of steps

Random Exploration

The behaviors of an RL agent constantly change throughout the training, starting from randomly exploring the environment to following the learned rules. Its strategy is influenced by how it initially explores the space. With different random seeds [1], an RL agent can develop different strategies to achieve the same goal under the same training settings. These strategies are discovered autonomously without being explicitly taught, and some of them even counter-intuitive. Fig 3.6 shows two solutions developed in *Avoidance*, trained with two different random seeds. Both two agents tend to move out of the canvas in the beginning and abruptly shift they directions to the right before moving out.

[1] A random seed specifies the start point of a random number sequence. Thus, different random seeds will cause the agent to explore the environment differently.

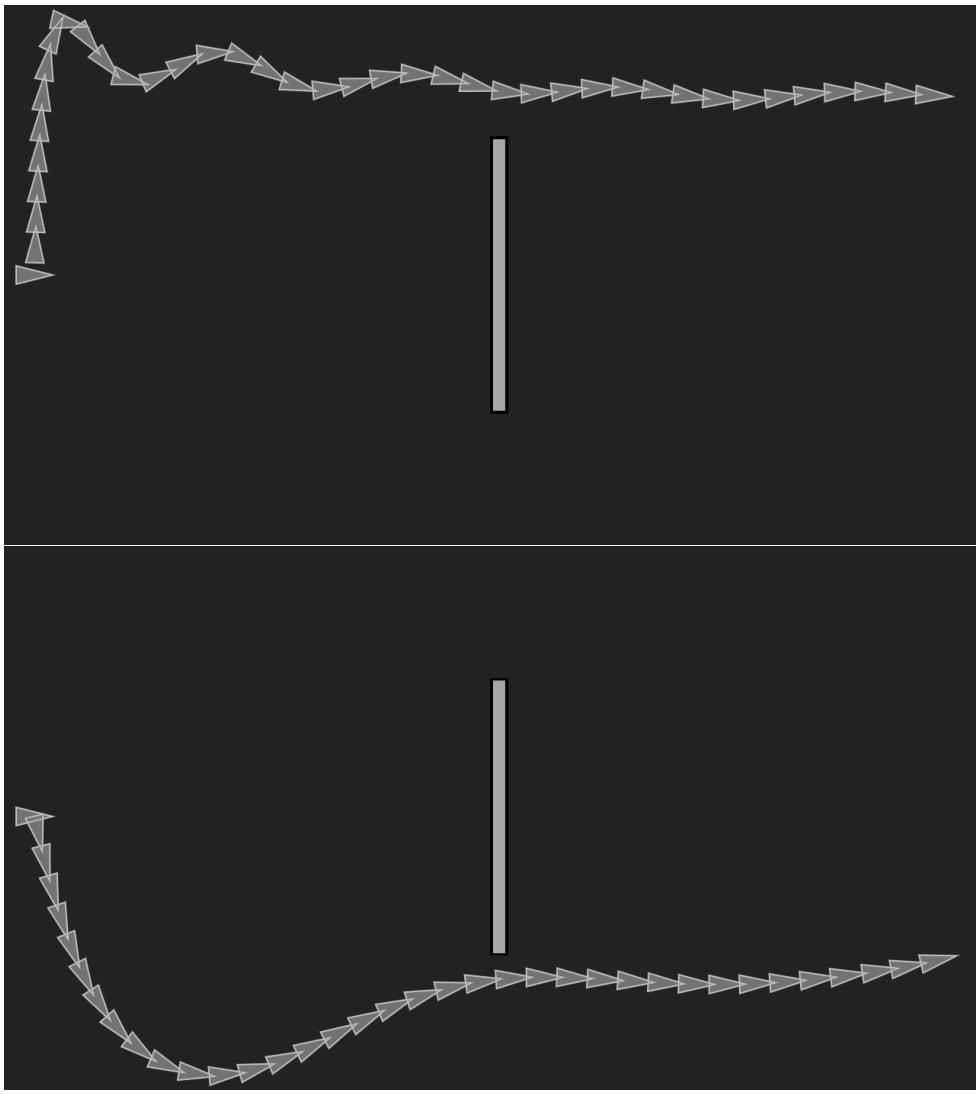


Figure 3.6: Footprints of two different strategies developed under the same training settings. The only variable is random seed.

State and Action Spaces

State spaces in reinforcement learning refer to how capable an RL-agent is to perceive its environment, and action spaces refer to how an RL-agent moves in its environment. The decisions on state & action spaces can largely influence the trained behaviors. An RL-agent that can observe the entire environment usually achieves the goal faster than an agent who only partially observers the environment, but a partially observable envi-

ronment may encourage the agent to explore some unconventional solutions. Regarding the action spaces, an agent obtaining full freedom to explore the environment may have a higher probability of avoiding local optimal than an agent who has limited freedom in action spaces. However, full freedom may also cause the agent to find a cheating solution.

In *Avoidance*, the state spaces are the current position of the agent, and the action spaces are a force to push the agent either to the top of to the bottom. At each step, the agent receives a constant force moving to the right. If the agent can explore both vertically and horizontally, it may evolve a behavior that moves to the left directly if the reward function returns negative values, or it may roam around its original if the reward function returns a constant positive value.

Reward function

In general, positive rewards encourage an agent to stay away from the terminal state as long as possible, whereas negative rewards encourage the agent to reach the terminal state as soon as possible. A group of reward functions were designed to explore emergent behaviors in *Avoidance*. The intention of those reward functions is the same as to reach to the right end without hitting the rectangle. Two of them are sparse reward functions, and the other two are shaped reward functions. The results are shown below.

- **Reward function:** The agent gets +1 reward every step before the terminal state. It get zero reward at the terminal state.

  Trained Behaviors: The agent abruptly turns when it approaches any of the edges, but it will eventually move out of the canvas.

- **Reward function:** The agent gets -1 reward every step before the terminal state. It get zero reward at the terminal state.

  Trained Behaviors: Depending on the exploration strategy, the agent may moves

directly to the rectangle or curly moves to the top or the bottom.

- **Reward function:** $reward = -distance(agent, right\_edge)$

  Trained Behaviors: The agent moves in zigzag to the right in the beginning and eventually moves out of the canvas from the top or the bottom edge. [2]

- **Reward function:** $reward = |agent.x - canvas\_width|$

  Trained Behaviors: Depending on the exploration strategy, the agent takes different curly movements to avoid the rectangle and reach to the right end.

3.3 Related Practice

In this section, the author's practices on agent-based generative art, deep-learning generative art, reinforcement learning, and their relation to RL-based generative art will be discussed. The practices explore various artistic experiments on generative art and technical improvements on reinforcement learning, which have been critical parts, conceptually and technically, to form the framework discussed before and to develop a tool to improve the accessibility of reinforcement learning, which will be discussed in the next chapter. All the works have been either exhibited or published at major conferences or galleries.

3.3.1 Agent-based Generative Art

This section introduces three agent-based generative art projects. They all present sequential movements of agents in either virtual or real space, based on a set of defined rules. *Anamorphic Fluid* creates a fluid physical model to move a group of images in a 3D

[2] It's the fastest way the agent discovered to reach the terminal state. In order to keep this reward function while achieving the goal, the environment can be modified to only terminate when the agent hits the right edge.

space. *Abstract Reality* and *Machinery Interference* use a modified Voronoi diagram for spatially positioning shapes and images in a 3D space. The emergent behaviors mostly take place at the interactions with the audience by using a camera to recognize bodies or faces. Since RL-based generative art is a subclass of agent-based generative art, the project can be naturally adapted to RL-based generative art by replacing the defined models to reward functions. The details of each project are described below.

Anamorphic Fluid

Completed year: 2015-2017

Type: Interactive animation

Collaborator: George Legrady, Donghao Ren

Anamorphic Fluid was realized as an artistic project to generate a continuously changing configuration in a simulated fluid environment where image-objects suspended in a liquid state are disrupted by external forces. The motivation was to create a human-computer interactive system that dynamically generates non-repeating structures in a virtual environment to allow observers to experience new perspectives that go beyond the original input images.

The overall workflow of the system can be seen in Fig 3.7. The elements in the system consist of a collection of pre-selected images that are suspended and floating in a virtual environment. The system switches between rest and motion states triggered by external forces, namely, the presence of people and their movements. The distinctions between the two states are reflected in the apparent differences in the movements and visual representations of the images as well as the overall acoustic effects in the environment. The system was programmed using Processing. In terms of hardware, the system consists of a display, a motion capture device, and a stereo audio output. The installation can be customized to include any number of images, sounds, and also the scale of visualization

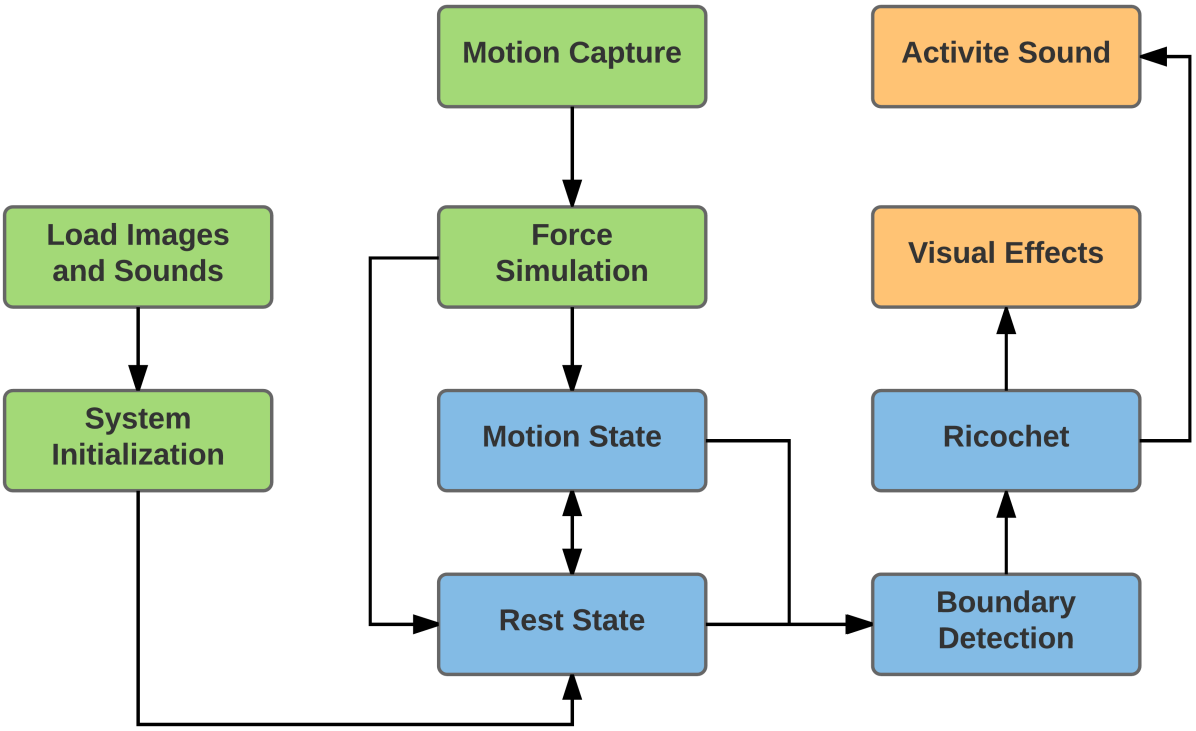


Figure 3.7: The system framework of *Anamorphic Fluid*

from screens to cinematic projection. An example of one version of the installation can be seen in Fig 3.8.

The system generates a dynamic 3D photomontage along with a combination of visual and audio effects. The most straightforward visual effect is the transformation of the images from rest to motion state when the presence and movements of users are detected. In rest state, the motion sensor detects no user and images are performing subtle floating motions based on rules of fluids at rest. In motion status, all the images are rapidly moving towards the user when the users presence is recognized. Images will also follow the hands moving trajectories in the virtual space as a flocking behavior. The invisible rectangular boundaries restrict the images movements by bouncing them back whenever any of the images collide against any of the boundary walls. In order to provide visual evidence of the collision, a flashing brightening and an increase in the color saturation

Figure 3.8: *Anamorphic Fluid* was premiered at the Edward Cella Art+Architecture Gallery in Los Angeles, USA to December 12th, 2015 until January 23rd, 2016.

are temporarily applied to the images at the hitting moment which then gradually fade-out. A sound randomly chosen from the pre-loaded sound database is triggered when the collision occurs, providing an audio indication of the collision.

Anamorphic Fluid was premiered at the Edward Cella Art + Architecture Gallery in Los Angeles, USA from December 2015 to January 2016. The system was mounted on the wall of the hallway leading to the main gallery that visitors passed through, inevitably activating the motion sensor. As visitors passed by, their movements activated image movements. Attracted by the effect, many viewers chose to return to explore and understand the mechanism of the system through their movements and presence. After a few minutes of observations, once the interaction of the piece became apparent, the audience would shift their focus to the content of the images.

The project was later exhibited at the Fellows of Contemporary Art Gallery in Los

Angeles, USA from January 2016 to March 2016, and then at the Dongdaemun Design Plaza Museum in Seoul, South Korea from August 2016 to September 2016. The images chosen for the three exhibitions include a combination of black and white photographs from a personal family archive from the late 1930s with contemporary night landscape scenes in color to highlight a mixed experience of past and present.

In the second iteration of the project, instead of pre-loading a group of images, the system constantly loaded images taken from a web camera, which was triggered by human faces. Once the images in the virtual environment reached to the maximum threshold, the upcoming images replaced the old ones. This version was presented at Currents New Media Festival in Santa Fe, New Mexico, USA in June, 2017.

Abstract Reality

Completed year: 2017

Type: Interactive installation

Abstract Reality is an interactive installation that creates 3D geometric art as an abstract expression of physical human bodies. The application takes viewers' physical features and their relation to the physical space as inputs to generate and place basic geometric forms in a virtual 3D space. Each geometric shape, virtual position and orientation, and color are affected by each viewer's physical positions, movements, and dominant colors. The overall structure of the geometric shapes is controlled by a modified Voronoi diagram, a computational geometric algorithm, to explore novel aesthetics. Fig 3.9 shows a collection of images generated by the system.

The idea behind the application is to connect the humans body and mind through indirect coordination. Inspired by Suprematism, which creates a world of only true reality, Abstract Reality project the physical world into a virtual world, where the contents are denoised and human-centered. The project was exhibited at the 2017 SIGGRAPH Asia

Figure 3.9: A series of images generated by the system of *Abstract Reality*. Each image represents a 3D abstract form of a person.

Art Gallery, Bangkok, Thailand. [69], as shown in Fig 3.10.

Machinery Interference

Completed year: 2017-2018

Type: Interactive Robotic Installation

Supervisor: George Legrady

Machinery Interference is a human-robot interactive installation that explores the relationship between the physical and the virtual through image-capture activities of an autonomous system consisting of a group of ground miniature robots. Each robot moves in a restricted area divided by five plastic boards that each board is painted in a unique color. All the boards are placed vertically on the ground based on Voronoi, a computational geometric algorithm, to construct a dynamic background for the robotic photographers. Fig 3.11 shows a conceptual image of the setup.

Figure 3.10: *Abstract Reality* was exhibited at 2017 SIGGRAPH Asia Art Gallery, Bangkok, Thailand.

The robots are actively searching for faces, and once they find one, they take a picture and send the image to a server from which the images are then retrieved and featured in a 3D virtual environment viewed on a 4K screen or a projector. All the photos are reassembled in the virtual space to form a continuously evolving 3D photomontage driven by the same algorithm applying on the plastic boards as an emergent aesthetic expression of the physical world. The audiences are encouraged to explore the space by presenting their faces in front of the robots, moving their bodies as obstacles to change the robots behaviors, or using their cell phones to participate in image-capture activities.

The project was invited by Times Arts Museum in Beijing, China, to participate in a group exhibition with Gary Hill, Daito Manabeand other seven international artists. In this exhibition, the boards were replaced by hundreds of cell phones sitting on different

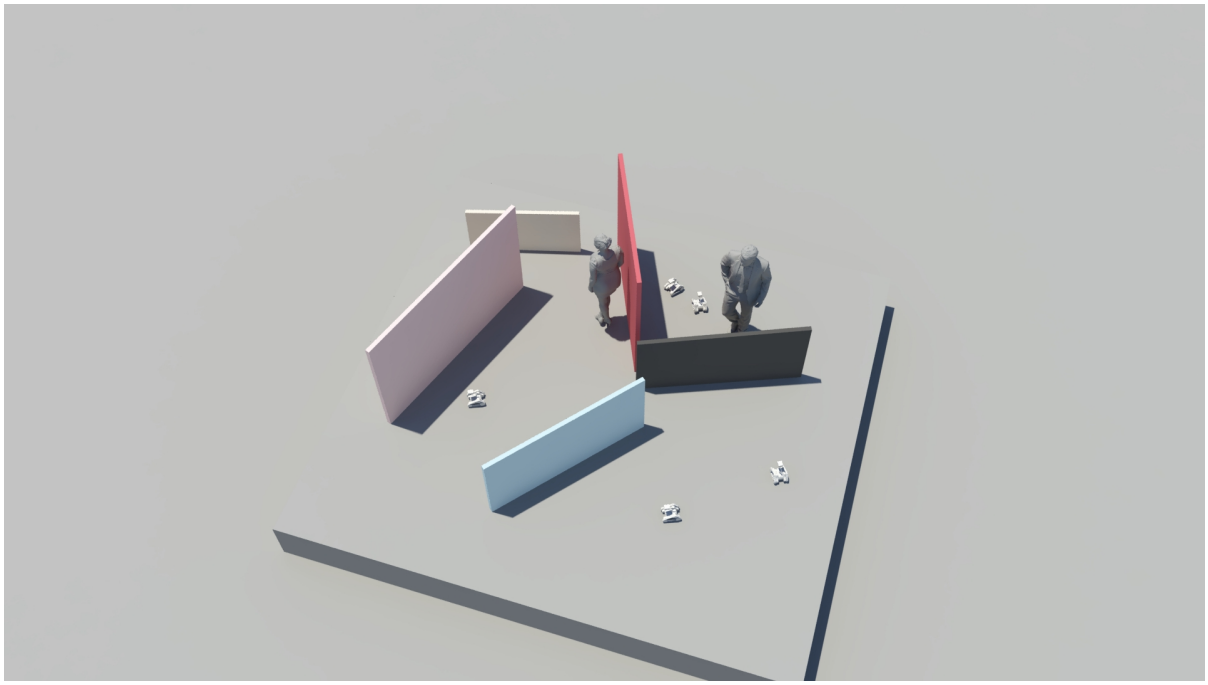


Figure 3.11: A conceptual image of *Machinery Interference* illustrating the setup discussed above.

lengths of poles. Fig 3.12 shows the setup in Times Arts Museum.

The project originated from a 10-week studio course directed by Professor George Legrady. The 18 students in the course were grouped into six teams tasked with assembling the robots. The course covered early camera history, the introduction of computational photography, computer vision, physical computing, wireless communication, and swarm intelligence. Teams also focused on aesthetic expression through the staging of the robot's active area, and designing the robotic behaviors, giving each robot a personality as they navigated around the objects in search of faces. The final project of the course was featured at the 2018 MAT End of Year Show, as shown in Fig 3.13.

Figure 3.12: *Machinery Interference* was exhibited at Times Arts Museum, Beijing, 2019

3.3.2 Deep-learning Generative Art

This section describes two practices on deep-learning generative art, aiming to go beyond the creative process introduced by Ahmed Elgammal. In *LAVIN*, a pre-trained neural network is used as an intermediate tool by inputting images and outputting texts describing the inputs. The texts then decide what participants can see in a virtual environment. In *Decomposition of Human Portraits*, the same neural network was used to generate texts to describe each input image. Besides, the project visualizes the saliency map of each input image. The creative processes in the two projects did not directly follow what Elgammal suggested, but the control of the deep learning model is still minimal. The details of each project are described below.

Figure 3.13: The original prototype of *Machinery Interference* was a group project from a studio course directed by Prof. George Legrady. The final project was exhibited at the 2018 MAT End of Year Show, University of California, Santa Barbara.

LAVIN

Completed year: 2019

Type: AI + VR Installation

Collaborator: Weidi Zhang

The current artificial intelligence (AI) technique allows a deep neural network to recognize over 20,000 different categories of objects. Meanwhile, countless deep neural networks are trained for different applications as the output from each neural network is a singular projection of the neural networks own understanding of the real world. Given the same visual input, for example, neural networks trained for facial recognition can only interpret the input as a collection of faces, while neural networks used for autonomous

driving can decompose the input into cars, pedestrians, trees, etc. Regardless of the complexity of the projection, it shapes the world value of the neutral network it belongs to. Therefore, an intriguing question arises as to what ground truth is in the modern AI age, given the fact that the most complex neural network model cannot inclusively represent the real world.

This virtual reality project tries to address this question by providing an immersive responsive experience to evoke peoples awareness regarding values and beliefs within the contexts of AI. Its also a novel attempt to connect virtual reality and AI, two current popular technical trends, for arts practice. The project consists of two major components: One is a neural network trained to recognize fewer than 100 objects that constantly analyzes the real world via a camera and outputs semantic interpretations. Specifically, the neural network chooses three objects to describe any given visual inputs. Since the neural network can only recognize a limited group of objects, it may falsely interpret an image that contains objects not in its training dataset, but it could never be able to realize the mistakes. The objects chosen to train the neural network, which are daily objects but not commonly observed, are carefully curated. The motivation is to trigger absurd results coming out from the neural network. The most recent result will be sent to a virtual reality environment discussed in the second component.

The second component involves the virtual reality environment consisting of the objects used to train the neural network, which are modeled based on photogrammetry and animated as 3D fluid abstract structures. All the objects are spatially organized in the virtual space based on their semantic relationships. Hammer and nails, for example, would be close to each other, but both of them might be far from umbrella or sunglasses. The real-time three-word description resulting from the neural network navigates the audience in the virtual space by transferring the person to the related objects. The experience in the virtual space is constantly responding to the surrounding environment in

the real world. Fig 3.14 presents the system diagram of *LAVIN*.

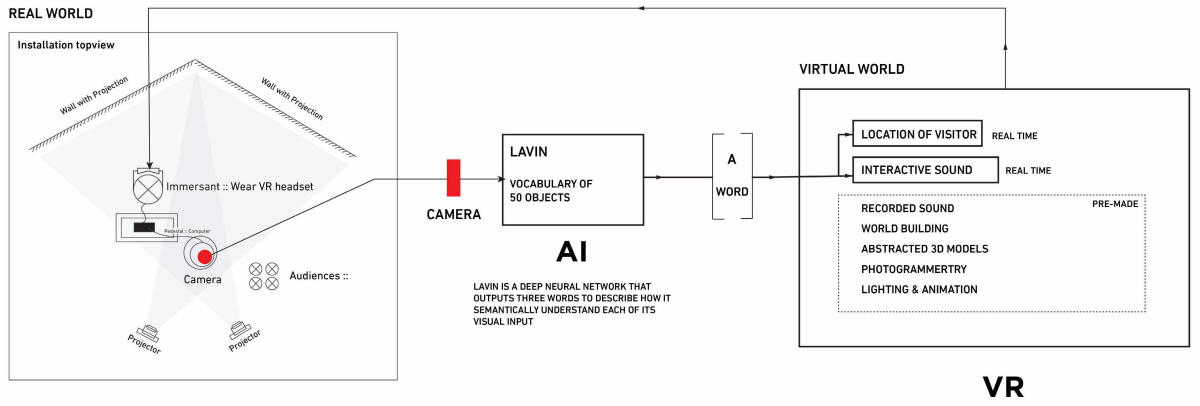


Figure 3.14: The system diagram of *LAVIN*.

The project was premiered at the 2019 SIGGRAPH Art Gallery in Los Angeles, CA, USA [70]. It was also featured at SWISSNEX in San Francisco, CA, USA in Nov 2019. Fig 3.15 shows the setup in the 2019 SIGGRAPH Art Gallery.

Decomposition of Human Portraits

Completed year: 2019

Type: Deep Neural Network Visualizations

Decomposition of Human Portraits is an interactive installation as a conceptual response to human digital identities in the modern artificial intelligence (AI) age. Once a human portrait is detected, the installation is triggered to take a photo and starts to analyze the photo by decomposing it into hundreds of square patches and sending them to a pre-trained deep neural network. The final outcome of each photo is a visualization of how the neural network visually interprets the portrait plus a generated three-word title of how the neural network semantically understands the portrait.

The current trend in AI is to use massive amounts of data and compute. In return,

Figure 3.15: *LAVIN* was premiered at SIGGRAPH 2019, Log Angeles, CA, 2019

machine learning algorithms have made remarkable progress in object recognition. Even though the performance is exceptional, the classification mechanism is still mysterious. *Decomposition of Human Portraits* aims to evoke awareness of the fragility of our digital identities managed under intelligent machines, by presenting how a reliable and well-trained system may falsely interpret human portraits. Fig 3.16 shows a collection of images generated by the system. The project was featured at the 2018 IEEE VIS Art Gallery, Berlin, Germany, and was also exhibited at 2019 ChinaVIS, Chengdu, Sichuan, China.

In *Decomposition of Human Portraits*, we used the technique introduced by Simonyan et al. [71] to create saliency maps of each input image. Saliency maps are a visualization

Figure 3.16: Ten paintings generated by the AI system behind *Decomposition of Human Portraits*.

technique which highlight areas in an image responsible for a given classification. Saliency maps provide insight into which regions of an input are responsible for a classification. In this project, the saliency maps are generated from back-propagation of a deep neural network trained on the ILSVRC-2013 dataset [3], which includes 1.2 million training images labelled into 1000 classes.

Specifically, we divided each input image into hundreds of square patches in the size of 224 pixels by 224 pixels and input each of those patches to the trained neural network for classification and back-propagation. Patches were recognized by the classifiers as castle, viaduct, Indian elephant, and others. The generated greyscale saliency maps were then colored based on the original inputs color scheme, and reassembled based on their location in the original portrait.

[3]http://www.image-net.org/challenges/LSVRC/2010/

3.3.3 Reinforcement Learning

This section presents two advanced techniques in reinforcement learning. One of the major trends in developing RL techniques is to improve training efficiency on high precision tasks. *Dynamic Experience Replay* augments replay buffers by prioritizing successful transitions generated during the training to improve training efficiency. *Guided Deep Deterministic Policy Gradient* improves training efficiency by combining a model-free RL algorithm and a model-based RL algorithm. The practices technically prepare the author for the development of an RL tool for non-experts. Several RL concepts, like replay buffer, are implemented in the tool.

Dynamic Experience Replay

Publication: Jieliang Luo and Hui Li, *Dynamic Experience Replay*, 2019 Conference on Robot Learning, 2019

Dynamic Experience Replay (DER) is a novel technique that allows Reinforcement Learning (RL) algorithms to use experience replay samples not only from human demonstrations but also successful transitions generated by RL agents during training and therefore improve training efficiency. It can be combined with an arbitrary off-policy RL algorithm, such as DDPG [24] or DQN [30], and their distributed versions.

The idea behind Dynamic Experience Replay (DER) is to augment human demonstrations using successful trajectories generated by RL agents during training, especially in cases where human demonstrations are not very helpful. We define *demonstrations* as either human demonstrations or the successful trajectories generated by RL agents. If DER is activated, regardless of the buffer structures mentioned above, each buffer allocates capacity $\mathbb{C}$ specifically for demonstrations. We refer to this as the *demonstration zone*. During training, all the successful episodes generated by RL agents are stored in a

pool. Periodically, each replay buffer randomly samples one successful episode from the pool and stores it in the demonstration zone. When the demonstration zone is full, the oldest transitions are discarded. DER's framework in a distributed architecture is shown in Fig. 3.17.

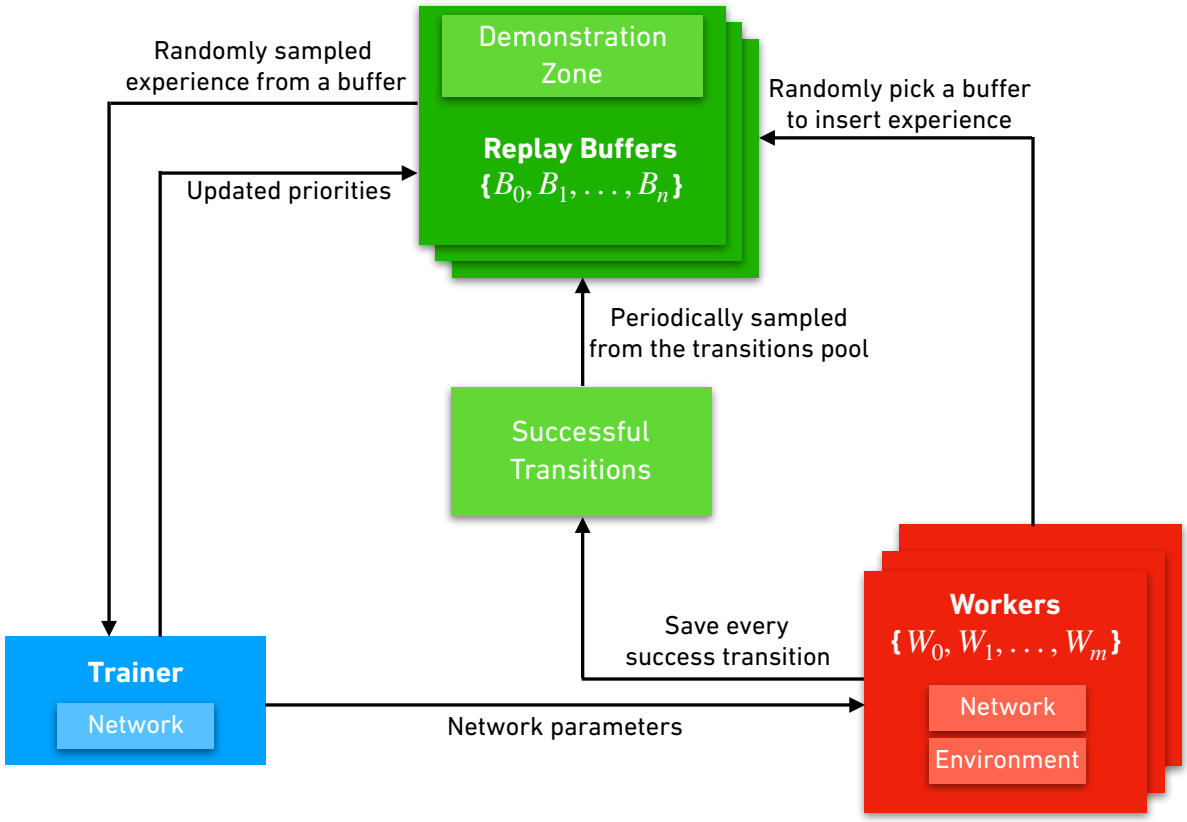


Figure 3.17: The Dynamic Experience Replay framework: multiple workers, each with its own instance of environment, and multiple replay buffers, each with capacity $\mathbb{C}$ for demonstrations. Human demonstration(s) are stored in the demonstration zones before training starts. During training, all successful transitions that are generated by workers are saved in a pool, which is sampled periodically by each replay buffer and stored in the demonstration zone.

The technique is initially built on Ape-X DDPG [72] and demonstrated on robotic tight-fitting joint assembly tasks, based on force/torque and Cartesian pose observations. In particular, experiments were conducted on two different tasks: peg-in-hole and lap-joint. In each case, we compare different replay buffer structures and how DER affects

them. Our ablation studies show that Dynamic Experience Replay is a crucial ingredient that either largely shortens the training time in these challenging environments or solves the tasks that the vanilla Ape-X DDPG cannot solve. We also show that our policies learned purely in simulation can be deployed successfully on the real robot.

Videos presenting the experiments are available at `https://sites.google.com/site/dynamicexperiencereplay`. The full paper can be seen in Appendix A.

Guided Deep Deterministic Policy Gradient

Publication: Yongxiang Fan, Jieliang Luo, and Masayoshi Tomizuka, *A learning framework for high precision industrial assembly*, In 2019 International Conference on Robotics and Automation (ICRA), pp. 811-817. IEEE, 2019.

This work proposed a learning framework called Guided Deep Deterministic Policy Gradient (Guided DDPG) for high precision assembly task. The framework contains a trajectory optimization and an actor-critic structure. The trajectory optimization was served as a semi-supervisor to provide initial guidance to actor-critic, and the critic network established the ground-truth quality of the policy by learning from both the semi-supervisor and exploring with policy gradient. The actor network learned from both the supervision of the semi-supervisor and the policy gradient of the critic. The involvement of critic network successfully addressed the stability issue of the trajectory optimization caused by the high-stiffness and the force/torque feedback.

The proposed learning framework, shown in Fig 3.18, constrained the exploration in a safe narrow space, improved the consistency and reliability of the model-based RL, and reduced the data requirements to train a policy. Simulation and experimental results verified the effectiveness of the proposed learning framework. Experimental videos are available at `https://www.decf.berkeley.edu/~yongxiangfan/ICRA2019/guidedddpg.html`.

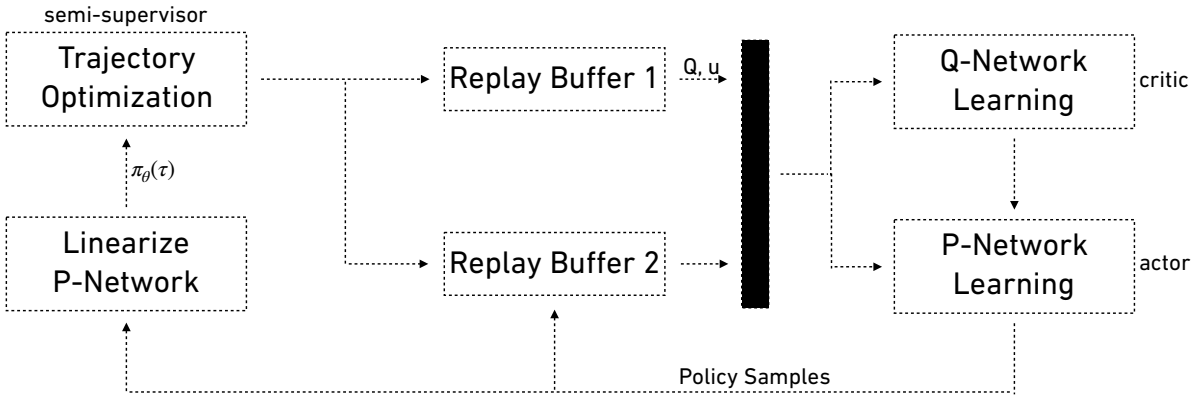


Figure 3.18: Illustration of the proposed learning framework (Guided DDPG). Trajectory optimization provides initial guidance to both actor and critic nets to avoid excessive exploration. The actor-critic nets gradually establish the evaluation system, instead of relying on pure supervised learning.

3.4 Other Achievements

In addition to the projects described in the previous section, I co-authored a book chapter on reinforcement learning with Sam Green in Koc Lab, UCSB which led me to consider the great potential of RL for artistic exploration. The chapter is called *Reinforcement Learning and Trustworthy Autonomy* [73], published at *Cyber-Physical Systems Security*. The chapter examines interpretable reinforcement learning in the context of Cyber-Physical Systems (CPS), which possess physical and software interdependence and are typically designed by teams of mechanical, electrical, and software engineers. The interdisciplinary nature of CPS makes them difficult to design with safety guarantees. When autonomy is incorporated, design complexity and, especially, the difficulty of providing safety assurances are increased. Vision-based reinforcement learning is an increasingly popular family of machine learning algorithms that may be used to provide autonomy for CPS. Understanding how visual stimuli trigger various actions is critical for trustworthy autonomy. In this chapter we introduce reinforcement learning in the context of Microsofts AirSim drone simulator. Specifically, we guide the reader through the

necessary steps for creating a drone simulation environment suitable for experimenting with vision-based reinforcement learning. We also explore how existing vision-oriented deep learning analysis methods may be applied toward safety verification in vision-based reinforcement learning applications. A shorter version of the chapter [74] was published at 2018 NVIDIA GPU Technology Conference (GTC).

Chapter 4

RL5: A p5.js Library for Reinforcement Learning

4.1 Overview

Stated in the previous chapter, reinforcement learning can open up new opportunities for a new type of generative art of which the rules in the system are not directly defined by artists but are learned by agent(s) interacting with its environments. However, reinforcement learning, unlike other generative models, is not accessible enough for non-experts. This chapter introduces RL5, a JavaScript library aiming to increase the accessibility of reinforcement learning by abstracting away technical details of reinforcement learning and allowing developers to focus on adapting reinforcement learning for artistic exploration.

The main purpose of RL5 is for rapidly prototyping RL environments and training RL policies in web browsers. It is built on top of p5.js so that developers can practice reinforcement learning in native p5.js language without any additional setups. That said, RL5 keeps users away from technical details, like callback and promise in JavaScript,

memory management in deep neural networks, or synchronization between training and rendering. The library combines RL algorithms and RL environments into one framework and offers simple APIs that developers can train and evaluate an RL agent in less than 20 lines of codes.

The library implemented three RL algorithms, Tabular Q-learning, REINFORCE, and DDPG, to cover all the three families of model-free RL. It also provides nine RL environments that three of them sever as an entry point for each algorithm respectively to help users to get started, and six others address autonomous agents in steering behaviors, which can be used as building blocks for complex systems. Fig 4.1 shows an overall structure of RL5.

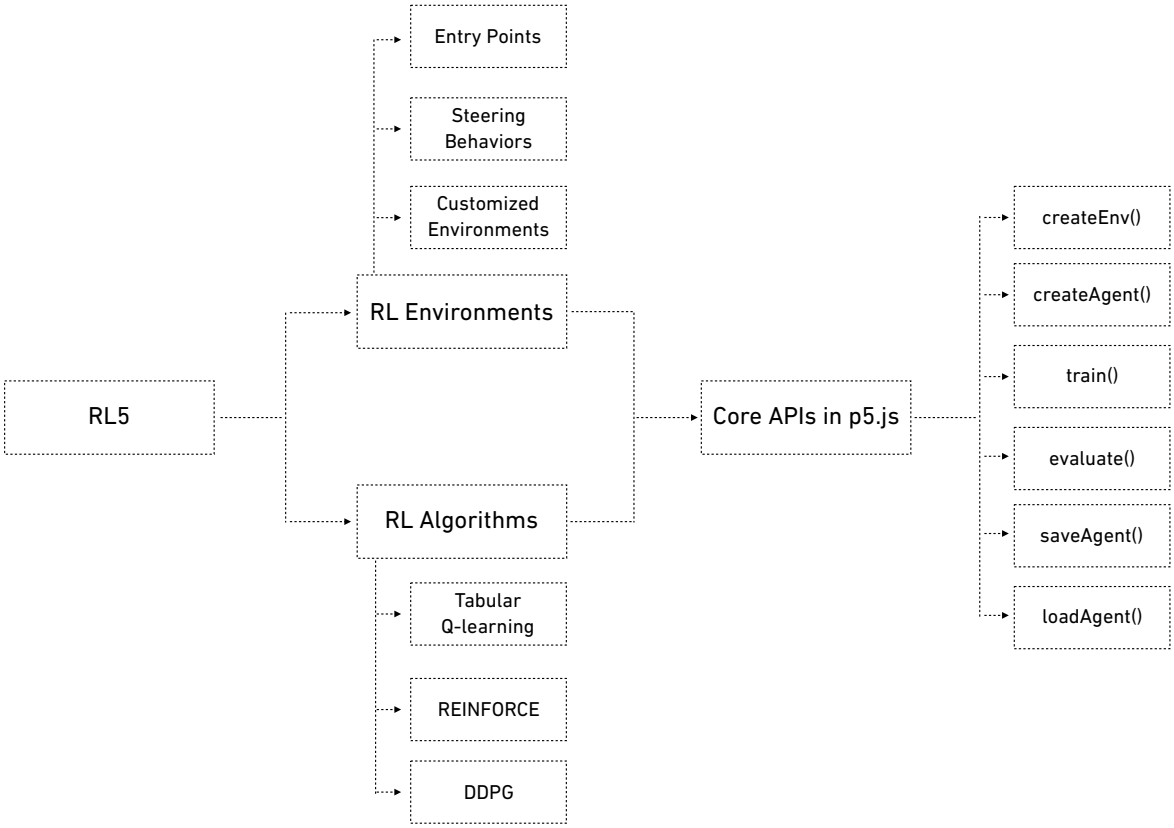


Figure 4.1: The overall structure of RL5

The remaining sections are structured as follows. Sec 4.2 explains in detail of the three implemented RL algorithms in RL5. Sec 4.3 describes the pre-defined RL environments in the library as well as a structure for customized RL environments. The core APIs of RL5 are introduced in Sec 4.4. Sec 4.5 demonstrates an example of using RL5.

4.2 RL Algorithms in RL5

Reinforcement learning can be divided into model-free methods and model-based methods. The difference is whether the agent has access to a dynamic model of the environment, which is usually a function to predict state transitions and rewards. RL5 focuses on model-free RL because it is, in general, easier to understand and tune than model-based RL.

Model-free RL algorithms are usually categorized into three families: Q-learning, policy gradients, and Q-function policy gradient. Thus, RL5 implements three RL algorithms to represent three categories, respectively. They are Tabular Q-learning [75], REINFORCE [76], and Deterministic Deep Policy Gradient (DDPG) [12]. An RL problem requires its environment to specify whether the state and action spaces are discrete or continuous and select a suited RL algorithm. Those algorithms cover all the combinations of different types of action and state spaces, as detailed in Table **??**, which provides a full spectrum for developers to design RL problems. The remaining parts of the section explain the core concepts of each algorithm.

4.2.1 Tabular Q-Learning

Q-learning is a model-free off-policy reinforcement learning algorithm. It's model-free because it solves the RL tasks directly using transitions from the environment without explicitly constructing a dynamic model of the environment. It's off-policy because the

	Action Space	**State Space**
Tabular Q-learning	Discrete	Discrete
REINFORCE	Discrete	Continuous
DDPG	Continuous	Continous/Discrete

Table 4.1: Three RL algorithms implemented in RL5 and the types of action and state spaces each algorithm supports.

learned value function directly approximates the optimal value function, independent of the policy being followed. The value function Q is defined by:

$$Q(s, a) = r(s, a) + \gamma \max_a Q(s', a) \tag{4.1}$$

where the Q-value is the immediate reward $r(s, a)$ at state s performing action a plus the highest Q-value possible from the next state s' preforming action a. γ is the discount factor indicating the importance of future rewards to the current state.

Since $Q(s', a)$ again depends on $Q(s'', a)$, it becomes a recursive equation that an update can be defined by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4.2}$$

where α is the learning rate.

Tabular Q-learning means the approximate value function is represented as a table, which is the simplest form for RL algorithms. Tabular Q-learning is suited for small and discrete state and action spaces, therefore it's an ideal entry-point RL algorithm. The pseudocode of tabular q-learning is stated in Algorithm 1.

The exploration and exploitation balance is controlled by the ϵ-greedy policy that the

Algorithm 1: Tabular Q-learning

```
Initialize a Q-table and set all Q-values to 0
while Q-table has not converged do
    Reset the agent and environment
    while Episode not terminated do
        Choose a from Q(s, *) using ε-greedy
        Take action a, observe r and s'
        Q(s_t, a_t) ← Q(s_t, a_t) + α[r_{t+1} + γ max_a Q(s_{t+1}, a) − Q(s_t, a_t)]
        s ← s'
Return trained Q-table
```

agent has a certain probability to randomly explore the environment regardless of the current policy.

In RL5, Tabular Q-learning was implemented only in native p5.js, which has a *Table* class to handle Q-table and its operations.

4.2.2 REINFORCE

REINFORCE is an on-policy algorithm under the policy gradients method of which the probabilities of actions leading higher return will be pushed up and the probabilities of actions leading lower return will be pushed down, until the optimal policy is reached.

Since Markov Decision Process often has stochastic state transitions and reward functions, the **objective** $J(\theta)$ of the agent is actually to maximize the *expected* sum of rewards under the **trajectory probability distribution** (defined in Eq. 4.8 below). This is achieved by discovering optimal policy parameters $\theta^\star$ for the objective function $J(\theta)$:

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau\sim p_\theta} \sum_{t=0}^{T-1} r(s_t, a_t) = \arg\max_\theta J(\theta), \tag{4.3}$$

where τ is the **trajectory** of state-action pairs $(s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ and p_θ is the trajectory probability distribution.

The REINFORCE method uses gradient ascent to adjust the policy parameters in a direction which increases $J(\theta)$ [76]. For notation convenience let $r(\tau) = \sum_{t=0}^{T-1} r(s_t, a_t)$, and by the definition of expectation, the objective can be written as:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta} r(\tau) = \int p_\theta(\tau) r(\tau) d\tau, \tag{4.4}$$

where $p_\theta(\tau)$ is the probability of a specific trajectory. Taking the gradient of $J(\theta)$ with respect to θ then gives:

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) r(\tau) d\tau = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau. \tag{4.5}$$

For reasons that will become clear, we recall the following identity:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = p_\theta(\tau) \nabla_\theta \log(p_\theta(\tau)), \tag{4.6}$$

allowing us to rewrite Eq. 4.5 as:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int p_\theta(\tau) \nabla_\theta \log(p_\theta(\tau)) r(\tau) d\tau, \\ &= \mathbb{E}_{\tau \sim p_\theta} \nabla_\theta \log(p_\theta(\tau)) r(\tau). \end{aligned} \tag{4.7}$$

We now explain why the identity in Eq. 4.6 was used. The probability of a sampled (i.e. experienced) trajectory τ has a probability that can be explicitly calculated only if the underlying state-transition function is known:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t), \tag{4.8}$$

where $p(s_0)$ is the probability of starting the trajectory in state s_0 and is independent

of θ, and $\pi_\theta(a_t|s_t)$ is the probability of the selected action given the state observation s_t. To better understand the notation $\pi_\theta(a_t|s_t)$, note that the policy is **stochastic**. In other words, when the policy is given a state observation s_t, the output of $\pi_\theta(s_t)$ is a vector of probabilities derived from the $softmax$ function[1]. In the discrete action-space environments considered here, there is one output probability per possible action. A random action is then drawn from the given probability distribution, and the probability of the selected action is denoted $\pi_\theta(a_t|s_t)$.

In real-world problems $p(s_{t+1}|s_t, a_t)$ is not known, so $p_\theta(\tau)$ would be impossible to calculate. However:

$$\begin{aligned} \log p_\theta(\tau) &= \log\big(p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)\big) \\ &= \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t), \end{aligned} \tag{4.9}$$

and replacing $\log p_\theta(\tau)$ in Eq. 4.7 with its expanded form gives:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim p_\theta} \nabla_\theta \Big[\log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)\Big] r(\tau), \\ &= \mathbb{E}_{\tau\sim p_\theta} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) r(\tau). \end{aligned} \tag{4.10}$$

In this form, we are able to approximate the gradient. Recall that π_θ is a neural network (or some other differentiable function), so the gradient of its log may be calculated explicitly given each a_t and s_t over the trajectory. Also, we know the sum of rewards $r(\tau)$ for each trajectory. Finally, the outer expectation is approximated by performing N

[1] $softmax(x_i|\boldsymbol{x}) := \frac{e^{x_i}}{\sum_{j=1}^{|\boldsymbol{x}|} e^{x_j}}$, where $\boldsymbol{x}$ is a vector of reals.

episodes, i.e. experiencing multiple trajectories, and then averaging the sums giving:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{n,t}|s_{n,t}) r(\tau_n). \tag{4.11}$$

After having obtained an approximation of the objective's gradient, we may use it to update the neural network parameters with standard stochastic gradient ascent:

$$\theta = \theta + \alpha \nabla_\theta J(\theta), \tag{4.12}$$

where α is the learning rate and whose appropriate value must be experimentally found.

The REINFORCE method works surprisingly well for a broad range of problems, and there are many improvements that have been made to it to increase its performance. Understanding the method presented here is a good foundation for approaching current literature. The pseudocode of REINFORCE is stated in Algorithm 2.

Algorithm 2: REINFORCE

Initialize θ arbitrarily
for *each episode* $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
  for $t = 1$ *to* $T - 1$ **do**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
return θ

REINFORCE trains a stochastic policy by sampling actions from the latest version of the policy. The exploration process is caused by the randomness in action selection, which gradually becomes less random over course of training. The downside is the policy might get trapped in local optima and miss the global optima.

In RL5, tensorflow.js was used to implement a neural network and handle its forward and backward operations. The main framework were implemented with native p5.js.

4.2.3 DDPG

Deep Deterministic Policy Gradient (DDPG) is an off-policy algorithm that concurrently learns a policy and a Q-function. Specifically, the Q-function is learned from the Bellman equation [77] like Q-learning with off-policy datasets, and the policy is learned from the Q-function. Therefore, it can be defined by:

$$a^*(s) = \arg\max_a Q^*(s, a) \tag{4.13}$$

where $a^*(s)$ learns an approximator and $Q^*(s, a)$ learns a separate approximator.

DDPG is off-policy because it uses an experience replay buffer that contains old transitions that might have been obtained from outdated policies. Experience replay [29] has been used to improve training efficiency in many RL algorithms, particularly for model-free RL, as it's less sample efficient than model-based RL. The technique has become popular after it was incorporated in the DQN [30] agent playing Atari games. Prioritized experience replay [31] is a further improvement to prioritize transitions so agents can learn from the most "relevant" experiences. Hindsight experience replay [32] stored every transition in the replay buffer not only with the original goal but also with a subset of other goals to acquire a more generalized policy. DDPG from Demonstrations [33] modified DDPG to permanently store a set of human demonstrations in the replay buffer to solve a group of insertion tasks.

DDPG is designed specifically for continuous action spaces as it uses a different method to compute $\max_a Q^*(s, a)$, which is usually exhaustively computed in discrete action spaces. The details of how DDPG calculates the max over actions will not be discussed here as it's beyond the scope of this dissertation. In general, it uses a target policy network to compute an action that approximately maximizes $Q_{\theta_{target}}$. The pseudocode of DDPG is stated in Algorithm 3.

Algorithm 3: Deep Deterministic Policy Gradient

Randomly initialize actor network μ with parameters θ, critic network Q with parameters ϕ, and an empty replay buffer $\mathbf{D}$

Copy the two networks parameters to two target networks respectively $\theta_{target} \leftarrow \theta$, $\phi_{target} \leftarrow \phi$

for *episode = 1, M* **do**

  Initialize a random process $\mathcal{N}$ for action exploration

  Get initial state s_1

  for t *= 1, T* **do**

    Observer state s_t

    Select action $a_t = clip(\mu_\theta(s_t) + \epsilon, a_{low}, a_{high})$, where $\epsilon \backsim \mathcal{N}$

    Execute action a_t and observe reward r_t and next state s_{t+1}

    Store transition(s_t, a_t, r_t, s_{t+1}) in $\mathbf{D}$

    Randomly sample a batch of transitions $\mathbf{N} = \{(s_t, a_t, r_t, s_{t+1})\}$ from $\mathbf{D}$

    Compute targets:

$$y(r_t, s_{t+1}) = r_t + \gamma Q_{\phi_{target}}(s_{t+1}, \mu_{\theta_{target}}(s_{t+1}))$$

    Update critic network by one step of gradient descent:

$$\nabla_\phi \frac{1}{|\mathbf{N}|} \sum_{(s_t, a_t, r_t, s_{t+1}) \in \mathbf{N}} (Q_\phi(s_t, a_t) - y(r_t, s_{t+1}))^2$$

    Update actor network by one step of gradient ascent:

$$\nabla_\phi \frac{1}{|\mathbf{N}|} \sum_{s \in \mathbf{N}} Q_\phi(s, \mu_\theta(s))$$

    Update two target networks:

$$\mu_{target} \leftarrow \tau \mu_{target} + (1 - \tau)\mu$$
$$\theta_{target} \leftarrow \tau \theta_{target} + (1 - \tau)\mu$$

In RL5, tensorflow.js was used to implement the two neural networks and their related operations. The main framework, replay buffer, action exploration were implemented with native p5.js.

4.3 RL Environments in RL5

In an RL problem, the RL environment is as equally important as the RL algorithm. As a browser-based library for small scale creative research in deep RL, RL5 provides a collection of RL environments as a foundation that developers could use those environments as building blocks to design and develop RL agents with creative goals and new behaviors. All the environments can be trained within minutes and rendered at 60 fps during the training, which serves the purpose of rapidly prototyping and testing RL ideas. Currently, the library provides nine environments in two categories as listed below:

- Entry Point (3)
    - 1D Grid World for Tabular Q-learning
    - CartPole for REINFORCE
    - Pendulum for DDPG
- Autonomous Agents (6)
    - Seek A Target (Discrete + Continuous)
    - Avoid An Obstacle (Discrete + Continuous)
    - Seek & Avoid (Discrete + Continuous)

The Entry Point category provides a classic example for each RL algorithm in RL5 as a starting point for users to get familiar with the concepts of RL algorithms and RL environments. The Autonomous Agent category introduces several steering behaviors in the context of reinforcement learning. The details of each environment are described below.

4.3.1 Entry Points

1D Grid World

Grid world is the most basic and classic problem in reinforcement learning. It is usually a 1D or 2D rectangular grid of size (N_x, N_y) that contains an agent starting from one grid and trying to find an optimal path to the target grid.

RL5 provides a 1D grid world in which a grey agent on the leftmost tries to reach the red target on the rightmost. It is one of the simplest RL environments and an ideal toy demo to begin reinforcement learning. Both the state spaces and the action spaces are discrete. Fig 4.2 shows the initial state of the environment rendered in p5.js. Below are the details of the environment in RL5.

Figure 4.2: Initial state of Grid World environment in RL5. The goal is to train the grey square to reach the red square. The grey square has no knowledge of the environment.

Reward Function: Reward is 0 for every step taken before the terminal step. Re-

State Spaces	Values
Index of Grid	$N \in [0, 9]$

Table 4.2: Discrete State Spaces of the 1D Grid World environment in RL5. This environment only contains 10 states, indexing from $state_0$ to $state_9$.

Action Spaces	Value
Move the black agent to the left	0
Move the black agent to the left	1

Table 4.3: Discrete Action Spaces of the 1D Grid World environment in RL5. The agent can choose either move to the left grid or the right grid.

ward is 1 at the terminal step.

Initial State: The agent always start at $state_0$ at the initial state.

Terminal Conditions: The agent reaches the red target.[2]

CartPole

CartPole is a classic control problem that was initially described by Barto, Sutton, and Anderson in 1983 [78], which shows "a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity."

The state spaces of CartPole are continuous and the action spaces are discrete. Even tho the environment only has two actions, pushing to the left and pushing to the right, the amount of velocity is not fixed as it also depends on the angle of the pole. Fig 4.3 shows one frame of CartPole during the training, rendered in p5.js. Below explains the details of CartPole in RL5.

[2]The agent bounces back if it hits the left boundary.

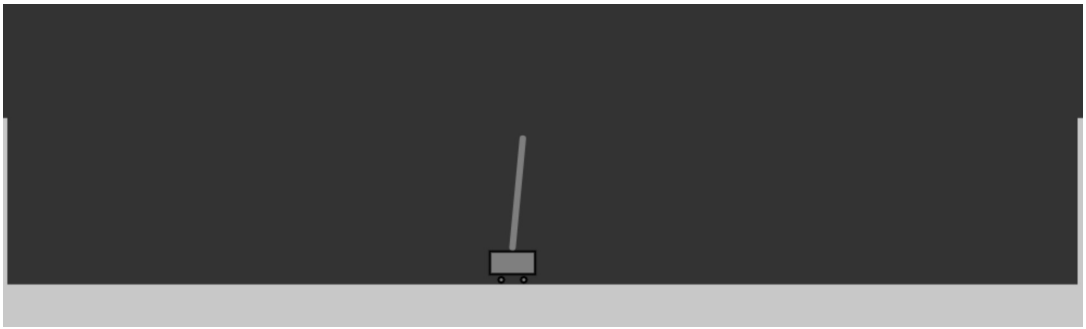


Figure 4.3: One frame of the CartPole environment in RL5. The goal is to train the cart to balance the pole by either moving to the left or to the right.

State Spaces	**Min**	**Max**
Cart Position	-4.8	4.8
Cart Velocity	-Infinite	Infinite
Pole Angle	-24 degree	-24 degree
Pole Velocity at Tip	-Infinite	Infinite

Table 4.4: Ranges of Continuous State Spaces of the CartPole environment in RL5

Action Sapces	**Value**
Push cart to the left	0
Push cart to the right	1

Table 4.5: Discrete Action Spaces of the CartPole environment in RL5

Reward Function: Reward is 1 for every step taken before the terminal step. Reward is 0 at the terminal step.

Initial States: All states are assigned a uniform random value in [-0.5, 0.5]

Terminal Conditions: (1) The center of the cart reaches the edge of the rendering area; (2) Pole angle is more than 12 degree; (3) The pole keeps balance for more than

500 steps.

Pendulum

The inverted pendulum is another classic tool in the control laboratories since the 1950s, originally used in linear control, like the stabilization of unstable systems [79], nonlinear control, like feedback stabilization [80], or control of chaotic systems [81]. Given the complexity and varieties, a pendulum has great potential to build a complex system in the contexts of generative art.

This section introduces a simple inverted pendulum swing-up problem that is suited for continuous action spaces RL algorithms, like DDPG. In the environment, a frictionless pendulum starts at a random position and the goal is to learn a strategy to swing up and stay upright, as shown in Fig 4.4. This example can be seen as a starting point to build more complex systems using pendulums, like double pendulum systems, that each pendulum can learn different behaviors with reinforcement learning. Both the state and action spaces are continuous in this environment. The details are listed below.

State Spaces	**Min**	**Max**
Cosine of Angle of Pendulum with Vertical	-1.0	1.0
Sine of Angle of Pendulum with Vertical	-1.0	1.0
Angular Velocity of Pendulum	-8.0	8.0

Table 4.6: Ranges of Continuous State Spaces of the Pendulum environment in RL5

Reward Function: The agent always get a negative reward until it gets upright. The precise reward function is defined by:

$$r = -(\theta^2 + 0.1 \times \omega^2 + 0.001 \times action^2)$$

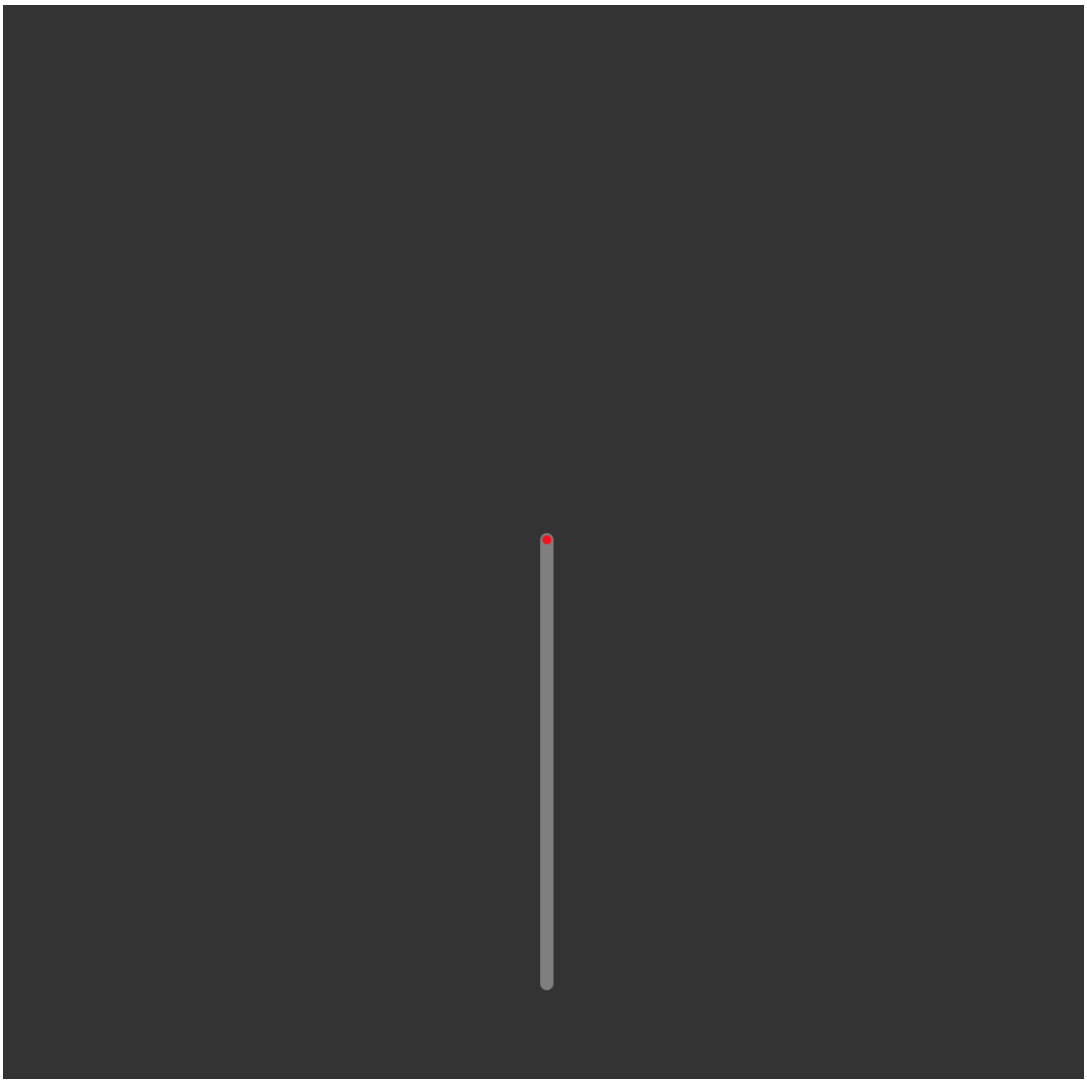


Figure 4.4: One frame of the Pendulum environment in RL5. The goal is to train the pendulum to swing up and keep standing

Action Spaces	**Min**	**Max**
Joint Effort	-2.0	2.0

Table 4.7: Ranges of Continuous Action Spaces of the Pendulum environment in RL5

where θ is the angle of the pendulum with vertical and ω is the angular speed.

Initial States: The initial angle is randomly assigned from $-\pi$ to π, and the angular

velocity is randomly assigned between -1 and 1.

Terminal Conditions: A maximum number of steps, usually defined by users.

4.3.2 Autonomous Agents

An autonomous agent is a common element in an autonomous system. Defined by Franklin and Graesser, "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [82] Namely, an autonomous agent makes its own choices about how to act in its environment.

In this section, the author introduces six RL environments addressing two steering behaviors, seek and avoid, proposed by Reynolds. Craig W. Reynolds proposed a group of steering behaviors [83] that can simulate life-like and improvisational manners for autonomous agents. Those agents, refereed by Reynolds as *situated agents* as they need to be situated in an environment, can be used as elementary building blocks for RL-based generative art. Instead of programming specific rules to simulate the steering behaviors, this section intends to create RL environments to train the behaviors. Specifically, six environments are created to cover three different scenarios, which are *seek a target*, *avoid an obstacle*, and *seek & avoid*. Each scenario covers both discrete and continuous action spaces. All the environments follow the same format, which can be easily extended to other steering behaviors. The next section will introduce how to define a customized RL environment in RL5.

Seek A Target

The goal of the Seek A Target scenario is for the agent to reach the target within the given maximum steps. In discrete action spaces, it can be seen as a massive 2D grid

world in which the agent moves UP/DOWN/LEFT/RIGHT. In continuous actions, the agent is controlled with a steering force adapted from Shiffman's implementation [63]. The screenshots of each environment are Fig 4.5 and Fig 4.6, respectively. The following describes the details of the environments.

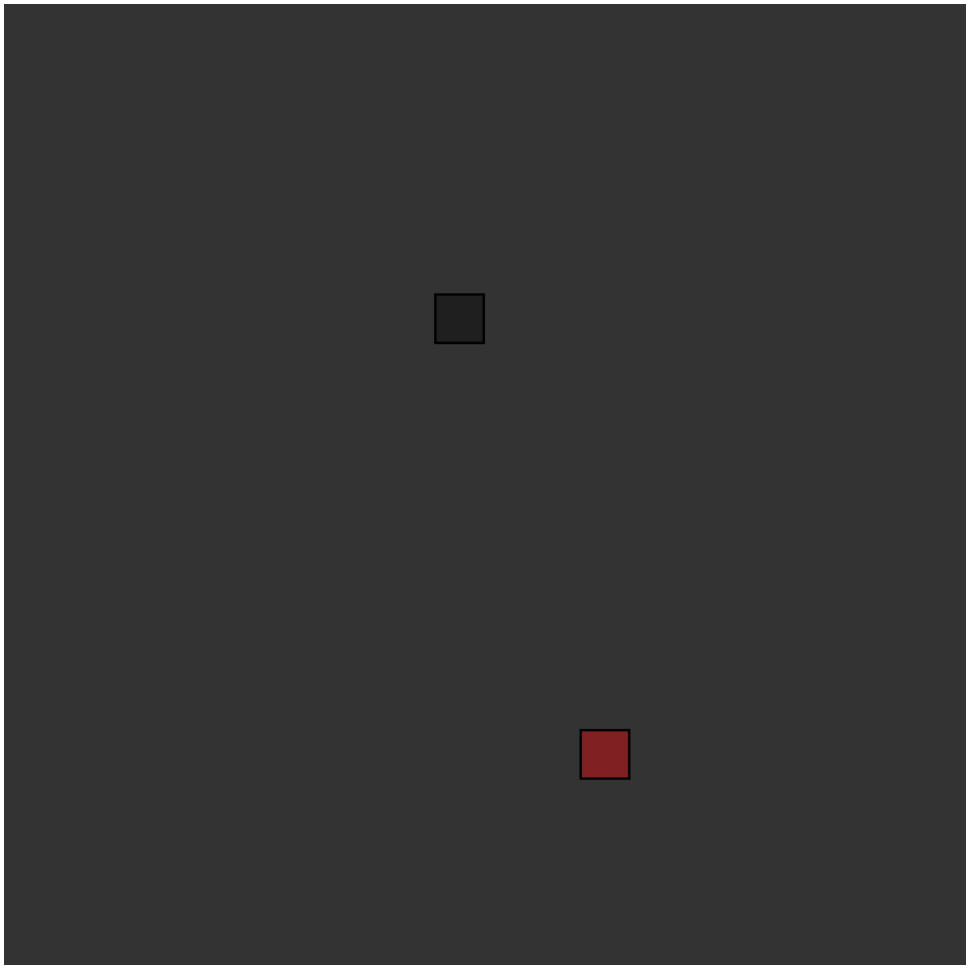


Figure 4.5: One initial state of Seek A Target in Discrete Action Spaces. The goal is to train the red square to reach the black square within the given maximum steps.

Reward Function: The agent gets zero rewards every step before it reaches the target. It gets one reward when it reaches the target.

Initial States: The initial position of the agent and the target are randomized within the size of the canvas.

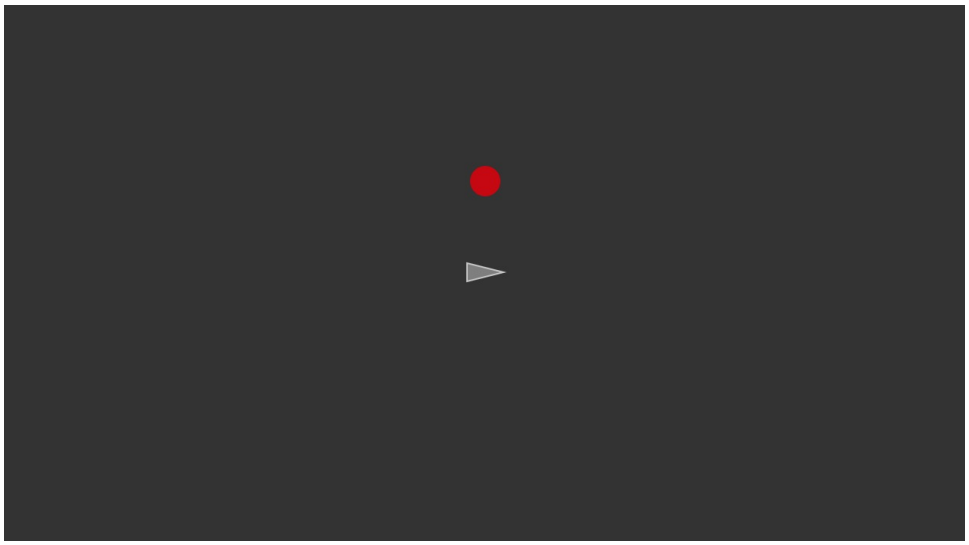


Figure 4.6: One initial state of Seek A Target in Continuous Action Spaces. The goal is to train the grey triangle to reach the red circle within the given maximum steps.

State Spaces	Min	Max
Normalized X Coordinate of the Agent	0.0	1.0
Normalized Y Coordinate of the Agent	0.0	1.0
Normalized X Coordinate of the Target	0.0	1.0
Normalized Y Coordinate of the Target	0.0	1.0

Table 4.8: Ranges of continuous state spaces of the Seek A Target environment in RL5, applying for both discrete & continuous action spaces.

Terminal Conditions: (1) The agent reaches the target; (2) The agent moves out of the canvas; (3) The episode reaches a maximum number of steps, usually defined by users.

Discrete Action Spaces	Value
Push the agent to the left	0
Push the agent to the right	1
Push the agent to the top	2
Push the agent to the bottom	3

Table 4.9: Discrete action spaces of the Seek A Target environment in RL5

Continuous Action Spaces	Min	Max
Force	-2.0	2.0

Table 4.10: Ranges of continuous action spaces of the Seek A Target environment in RL5

Avoid An Obstacle

Avoid obstacles is another classical problem in navigation. RL5 describes a scenario in which an agent moves from left to right and tries to avoid a rectangular obstacle in the middle of the way. At every step, the agent receives a constant horizontal force to move it to the right. Similar to Seek A Target, it also has two versions in terms of action spaces. The screenshots of each environment are Fig 4.7 and Fig 4.8, respectively. The following describes the details of the environments.

State Spaces	Min	Max
Normalized X Coordinate of the Agent	0.0	1.0
Normalized Y Coordinate of the Agent	0.0	1.0

Table 4.11: Ranges of continuous state spaces of the Avoid An Obstacle environment in RL5, applying for both discrete & continuous action spaces.

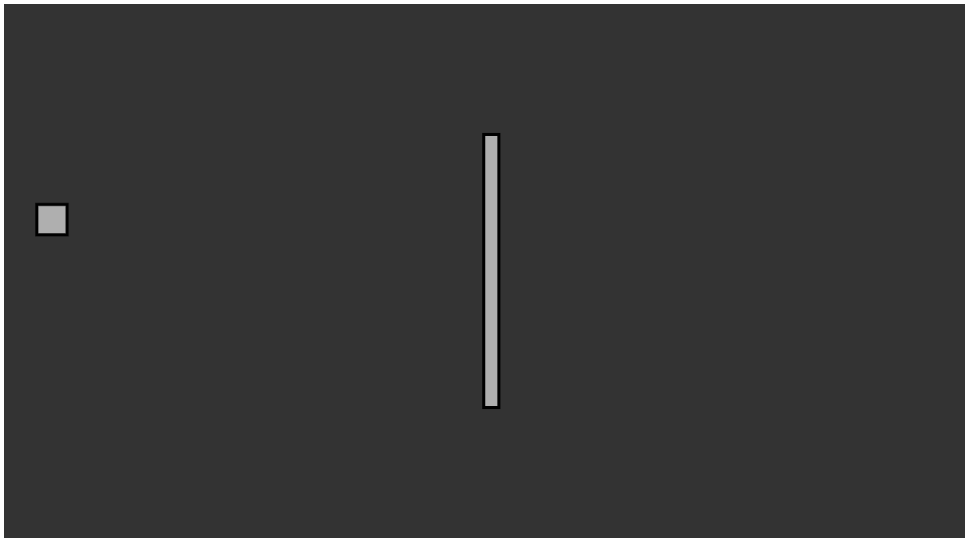


Figure 4.7: One frame of Avoid An Obstacle in Discrete Action Spaces. The goal is to train the square to move to the right without hitting the rectangle.

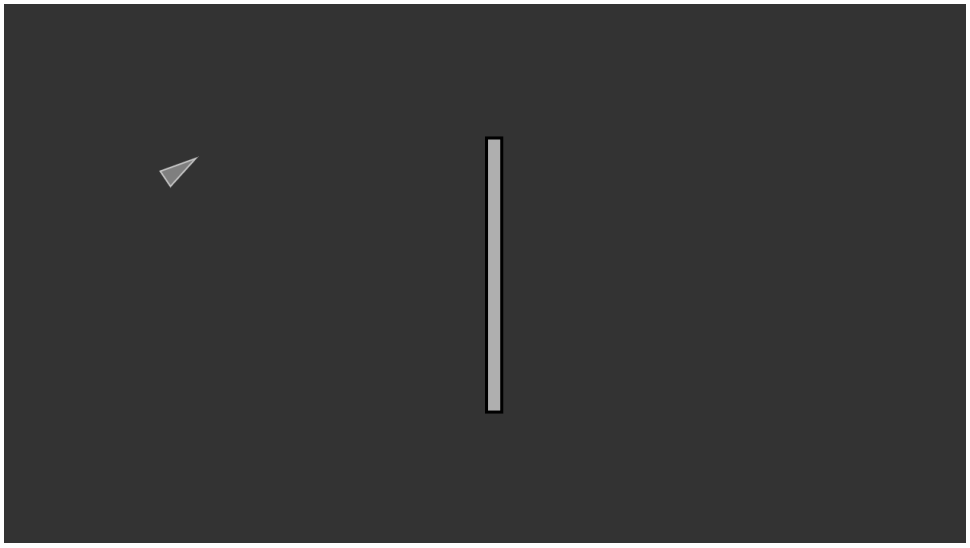


Figure 4.8: One frame of Avoid An Obstacle in Continuous Action Spaces. The goal is to train the triangle to move to the right without hitting the rectangle.

Reward Function: The agent gets -1 reward every step before the terminal state. It gets zero rewards at the terminal state.

Initial States: The X-coordinate of the agent is fixed. The Y-coordinate is randomized within the height of the canvas.

Discrete Action Spaces	Value
Push the agent to the top	0
Push the agent to the bottom	1

Table 4.12: Discrete action spaces of the Avoid An Obstacle environment in RL5

Continuous Action Spaces	Min	Max
Horizontal Force	-2.0	2.0

Table 4.13: Ranges of continuous action spaces of the Avoid An Obstacle environment in RL5

Terminal Conditions: (1) The agent hits the obstacle; (2) The agent reaches one of the boundaries; (3) The episode reaches a maximum number of steps, usually defined by users.

Seek & Avoid

Seek & Avoid combines the two previously discussed behaviors into one scenario in which an agent needs to reach a target while to avoid hitting obstacles. Likewise, it has a discrete-action version and a continuous-action version. This scenario shows the adaptability of RL environments in RL5 that individual steering behaviors could be added up to build a complex system without reconstructing the entire environment. The screenshots of each environment are Fig 4.9 and Fig 4.10, respectively. The following describes the details of the environments.

Reward Function: The agent gets -1 reward every step before the terminal state. It gets 10 rewards at the terminal state if it reaches the target and -10 rewards if it hits one of the obstacles.

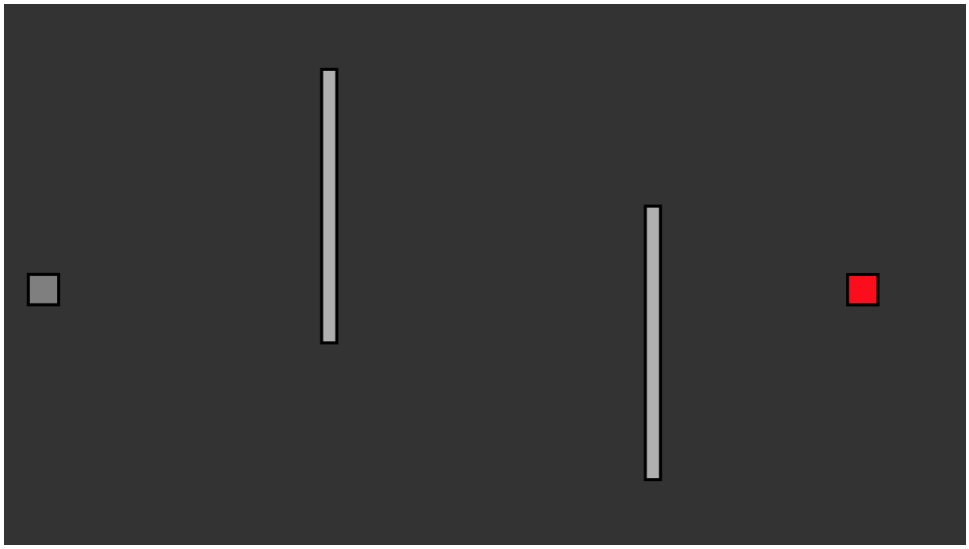


Figure 4.9: One frame of Seek & Avoid in Discrete Action Spaces. The goal is to train the grey square to reach the red square without hitting the two rectangles.

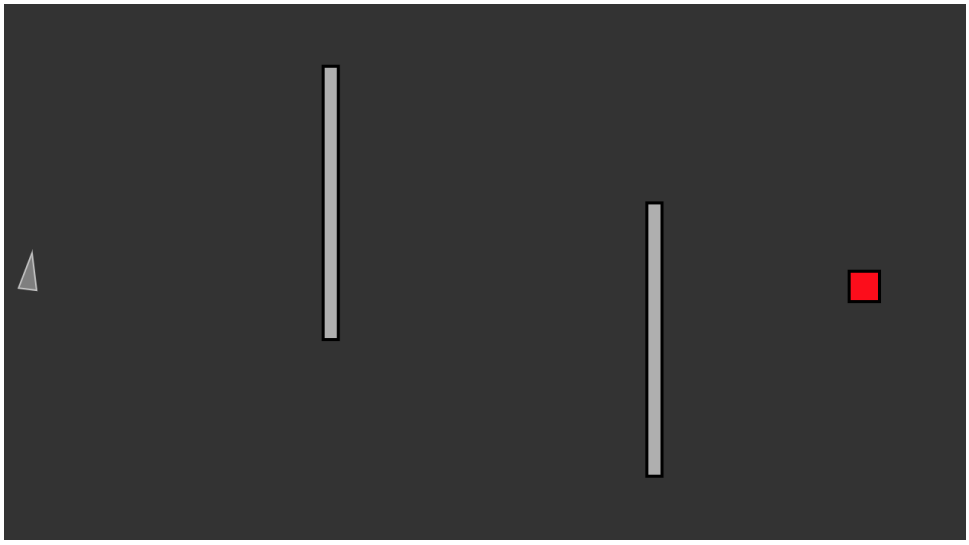


Figure 4.10: One frame of Seek & Avoid in Continuous Action Spaces. The goal is to train the grey triangle to reach the red square without hitting the two rectangles.

Initial States: The X-coordinates of the agent and the target are fixed. The Y-coordinates are randomized within the height of the canvas.

Terminal Conditions: (1) The agent hits the obstacle; (2) The agent reaches one of the boundaries; (3) The agent reaches the target; (4) The episode reaches a maximum

State Spaces	Min	Max
Normalized X Coordinate of the Agent	0.0	1.0
Normalized Y Coordinate of the Agent	0.0	1.0
Normalized X Coordinate of the Target	0.0	1.0
Normalized Y Coordinate of the Target	0.0	1.0

Table 4.14: Ranges of continuous state spaces of the Seek & Avoid environment in RL5, applying for both discrete & continuous action spaces.

Discrete Action Spaces	Value
Push the agent to the left	0
Push the agent to the right	1
Push the agent to the top	2
Push the agent to the bottom	3

Table 4.15: Discrete action spaces of the Seek & Avoid environment in RL5

Continuous Action Spaces	Min	Max
Force	-2.0	2.0

Table 4.16: Ranges of continuous action spaces of the Seek & Avoid environment in RL5

number of steps, usually defined by users.

4.3.3 Define A RL Environment

RL5 allows users to define their own RL environments in native p5.js. In RL5, a RL environment is broken down into the following components: *constructor*, *reset*, *step*,

isDone, *reward*, *getState*, *render*, and *randomWalk*. The following list elaborates each components:

- **constructor**: initialize constants and variables of the environment; reset the environment.
- **reset**: reset the environment to its initial states in order to start a new training episode. The initial states could be either fixed or randomized within a certain range.
- **step**: define transition dynamics to move the agent in the environment based on the given action; return a numeric reward and if the terminal state has been reached.
- **isDone**: define terminal states and determine if any of the terminal states has been reached.
- **reward**: define reward function and return a numeric value after each step.
- **getState**: return the states of each step.
- **render**: visualize all the elements in the environment.
- **randomWalk** (optional): randomly move the agent based on the action spaces. The purpose is to test if the environment is defined properly.

The structure is designed to be algorithm-agnostic. Typically, a simple RL environment in RL5 is less than 150 lines of code. Fig 4.11 shows the abstraction of RL environments in RL5, implemented in Sublime[3].

[3] `https://www.sublimetext.com/`

```
class EnvName {
    constructor(){
        this.reset();
    }

    reset(){}

    step(action){
        const info = [this.reward(), this.isDone()];
        return info;
    }

    isDone(){}

    reward(){}

    getState(){}

    render(){}
}
```

Figure 4.11: The abstraction of RL environments in RL5, implemented in Sublime

4.4 Core APIs of RL5

This section introduces three major components of the APIs in RL5, which are *setup an environment and agent*, *train and evaluate an RL policy*, and *access trained agents*.

4.4.1 Setup An Environment and Agent

Defining an RL environment and selecting a suitable RL algorithm is usually the starting point of reinforcement learning. In every step during the training, the algorithm takes states from the environment and outputs actions to the environment. Since states and actions could be either discrete or continuous, RL algorithms are not always inter-

changeable. DDPG, for example, does not work for most of the grid world environments as the algorithm does not support discrete actions.

As mentioned earlier, RL5 provides three RL algorithms covering all the combinations of different types of state and action spaces and nine pre-defined RL environments. `createEnv()` is for initializing a pre-defined or a customized RL environment by giving a name. `createAgent()` is for initializing an RL agent by giving the name of the selected RL algorithm, the RL environment, and hyper-parameters[4]. Those two functions complete the setup of the RL environment and the RL agent, and should be called in `setup()` of p5.js. The structure of the initialized table or neural network(s) is printed in the JavaScript console. Fig 4.12 shows a neural network's structure of a REINFORCE agent.

```
_________________________________________________________________
Layer (type)                 Output shape              Param #
=================================================================
dense_Dense1 (Dense)         [null,10]                 50
_________________________________________________________________
dense_Dense2 (Dense)         [null,2]                  22
=================================================================
Total params: 72
Trainable params: 72
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.12: The neural network structure of a REINFORCE agent in RL5. It has one hidden layer with ten neurons and output two actions.

Hyperparameters are grouped as one constant in the dictionary data structure. It is unnecessary to declare all the hyper-parameters of one algorithm as they all have default

[4]TensorFlow.js is required if REINFORCE or DDPG is used. Currently, RL5 has been tested on TensorFlow.js version 1.0.0.

values defined in each algorithm.

4.4.2 Train & Evaluate A RL Policy

In RL5, `train()` trains a RL agent and `evaluate()` evaluates the agent. Both of them take the created agent as the only variable. `train()` has to be an async function in JavaScript because of the back-propagation process. In order to render the training process at 60 fps in browsers, `train()` is designed to be positioned in `setup()` of p5.js. On the other hand, `evaluate()` should be placed in `draw()` function of p5.js.

An ending condition needs to be defined in the hyper-parameter constant, otherwise the training keeps going. An ending condition consists of two keys, *criterion* and *condition*. RL5 initially supplies four criteria, *steps*, *episodes*, *rewards*, and *successes*. *condition* takes one of the following three values: *more*, *less*, and *equal*. Here is an example to define an ending condition to end the training after the agent lasts more than 700 steps in one episode:

```
endingCondition: {
    "steps": 700,
    "condition": "more"
}
```

Listing 4.1: Define an ending condition of which training ends after the agent lasts more than 700 steps in one episode

The real-time progress of training and evaluation can be monitored from the JavaScript console. Fig 4.13 shows the training progress of training a DDPG agent in the Avoid An Obstacle environment.

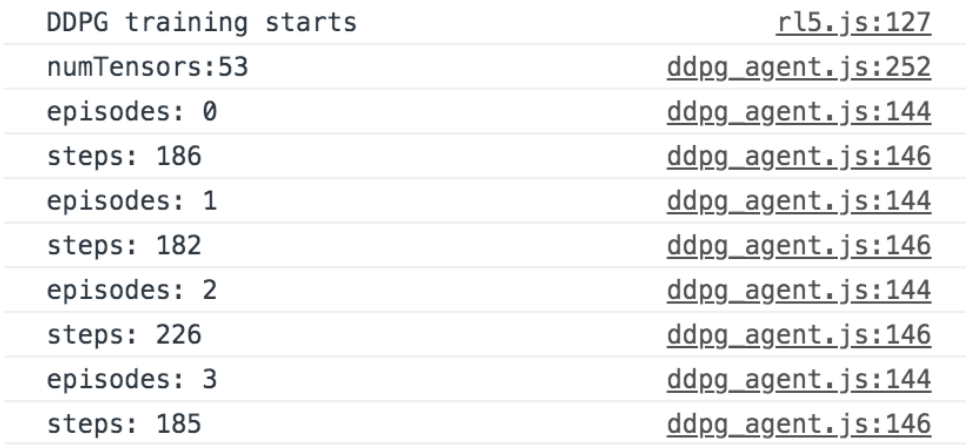


Figure 4.13: A screenshot shows the progress of training a DDPG agent for the Avoid An Obstacle environment.

4.4.3 Access Trained Agents

A key to make it trivial for creators to use reinforcement learning is to reuse and combine trained RL agents in different environments. `saveAgent()` and `loadAgent()` enable developers to save a trained agent in IndexedDB and load it to another p5.js environment. It also supports loading multiple agents simultaneously to one environment. Developers can use `checkSavedModels()` to see all the saved models in their IndexedDB storage and use `checkOneModel()` to see the details of a particular model.

A list of all the APIs of RL5 and their descriptions is maintained at `https://rodgerluo.github.io/RL5/reference/reference.html`.

4.5 An Example of RL5 - Avoid An Obstacle

This section describes how to train and evaluate a DDPG agent in the Avoid An Obstacle environment using RL5. Like any typical p5.js project, it consists of an html page to load all the dependencies and a JavaScript page to host the main codes. The entire codes of the example are shown in Fig 4.14. The core codes regarding initialization,

training, and evaluation are only from line 17 to line 35 in the JavaScript page. With better formatting, the core codes can be reduced to 11 lines.

```
1   const policyName = "DDPG";
2   const envName = "AvoidObstaclesContinous-v0"
3   const render = true;
4
5   let env;
6   let agent;
7
8   const config = {
9       stateSize: 2,
10      actionSize: 1,
11      endingCondition: {
12          "steps": 700,
13          "condition": "more"
14      }
15  }
16
17  function setup(){
18      createCanvas(640, 360);
19
20      env = rl5.createEnv(envName);
21      agent = rl5.createAgent(policyName, env, config);
22
23      rl5.train(agent);
24  }
25
26  function draw(){
27      background(51);
28
29      if(agent.stopTraining){
30          rl5.evaluate(agent);
31      }
32      if(render){
33          env.render()
34      };
35  }
```

Figure 4.14: The JavaScript page that trains and evaluates a DDPG agent in the Avoid An Obstacle environment.

The program can be run in any modern browser. When training begins, a browser window renders the real-time process and the JavaScript console shows training statistics.

At the beginning of the training, the agent has no knowledge of its environment, so it takes random actions and always moves out of the canvas before it even gets close to the obstacle. Fig 4.15 shows a moment that the agent was about to move out of the canvas from the bottom-left corner. In the middle of the training, the agent figures out that moving to the right is the general goal but hasn't developed a behavior to avoid the obstacle. Fig 4.16 shows a moment that the agent was about to hit the rectangle. However, the behaviors have changed from moving to the bottom to moving to the right. At the end of the training (the ending condition was the agent moves more 700 steps in one episode), the agent develops a strategy to avoid the obstacle while moving to the right end. Fig 4.17 shows that the agent passes the obstacle from its bottom side. The whole training process took 127 episodes, about 6 minutes.

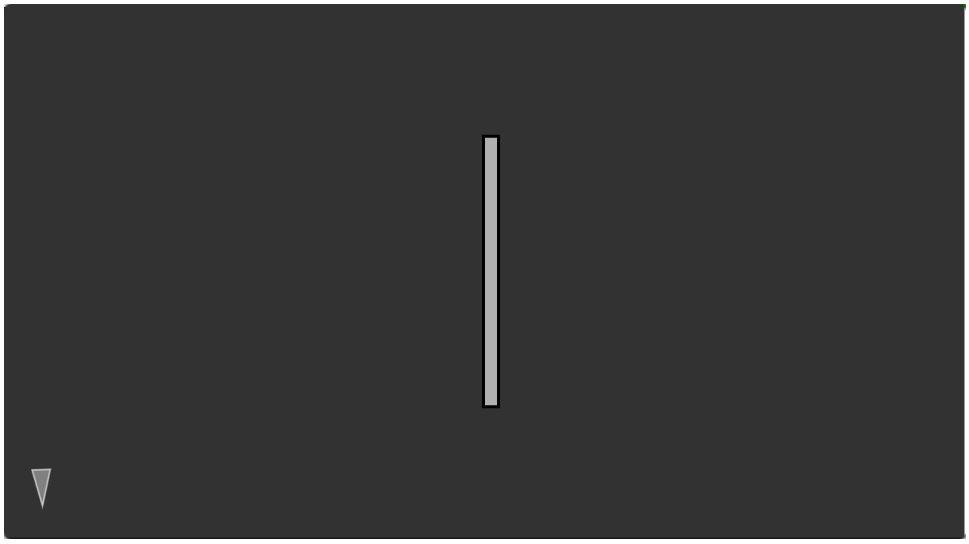


Figure 4.15: A screenshot shows a barely trained agent is about to move out of the canvas from the bottom-left corner.

Regarding emergent behaviors, by feeding different random seeds, the agent develops different methods to pass the obstacle. Fig 4.18 shows four different strategies to pass the obstacle. Such learned behaviors are more organic and unpredictable than rule-based behaviors. The videos of the whole training process and the different developed methods

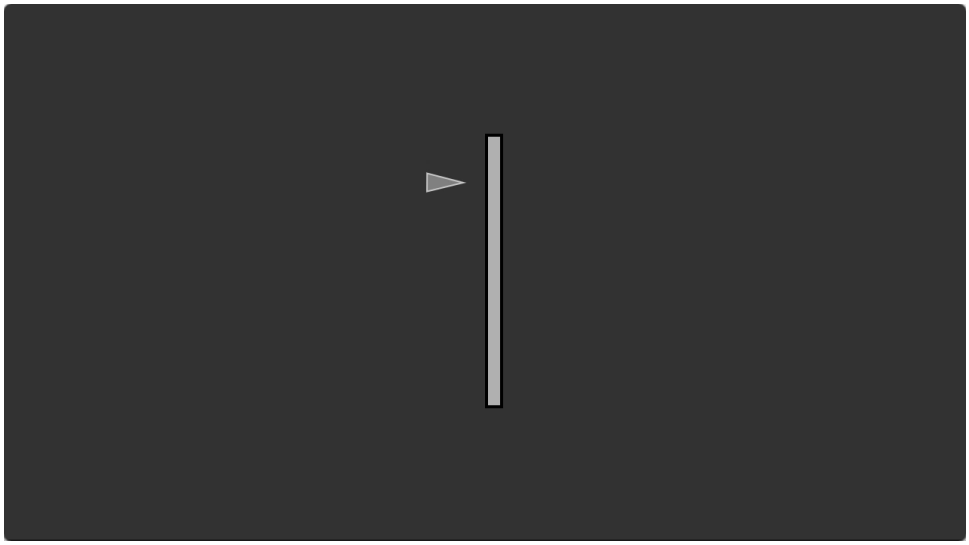


Figure 4.16: A screenshot shows a semi-trained agent is about to hit the obstacle. It figures out that moving to the right is the general goal but has not developed a behavior to avoid the obstacle.

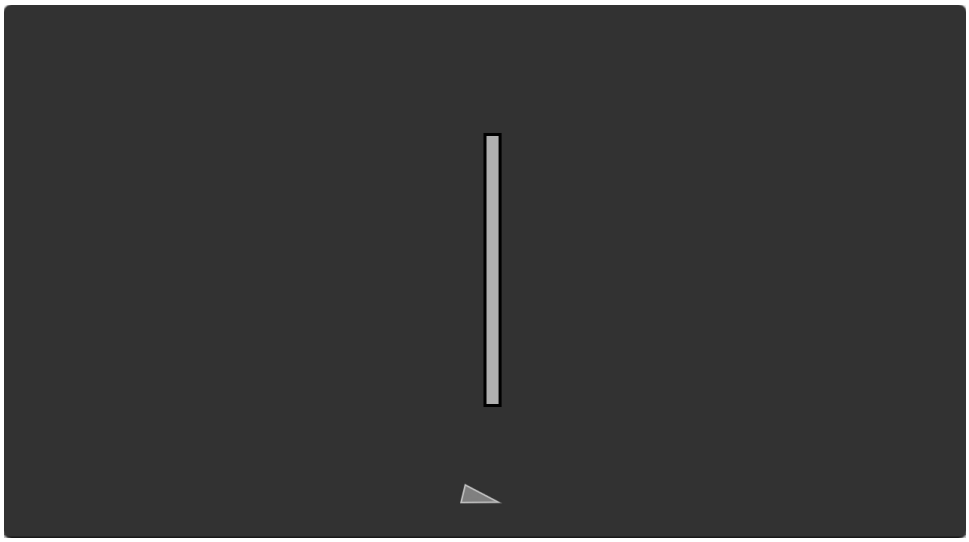


Figure 4.17: A screenshot shows a trained agent passes the obstacle from its bottom side.

can be seen from `https://vimeo.com/384560973`.

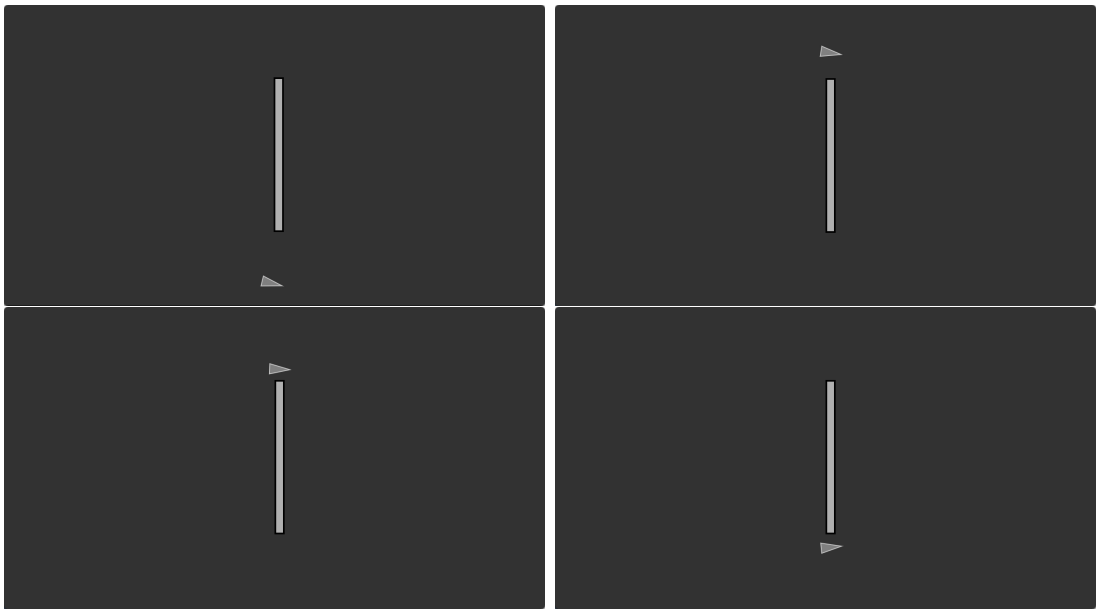


Figure 4.18: The same agent develops four different methods to pass the obstacle with four different random seeds.

Chapter 5

Applications of RL5

This chapter describes two pedagogical applications and two artistic applications of RL5. The pedagogical applications focus on understanding RL concepts and frame RL problems in the context of creativity rather than introducing technical details of reinforcement learning. The first pedagogical application was a 105-minute workshop given at the 2018 SIGGRAPH Asia Courses Program. The second one was a five-day workshop given at the College Art and Design at Beijing University of Technology. The artistic applications focus on exploring new opportunities of RL-based generative art. *Machine Learning with Visual Overlay* treats an RL agent as a control to send training-based messages to a shader program for visual renderings. *Action Through Inaction* uses an RL agent to conceptually represent an ancient Chinese philosophical concept in Taoism and explores novel aesthetics generated by the agent.

5.1 Pedagogical Applications

5.1.1 SIGGRAPH Asia 2018 Courses

> *For more than 30 years SIGGRAPH Courses have been the premier source for practitioners, developers, researchers, scientists, engineers, artists, and students who want to learn about the state-of-the-art in computer graphics and related topics. At SIGGRAPH Asia 2018 hundreds of visitors will attend its courses to broaden and deepen their knowledge and to learn the secrets of new directions. The SIGGRAPH Asia 2018 Courses program will aim to push the boundaries of Computer Graphics and Interactive techniques, by providing master classes by leading international experts from academia and industry, and also by broadening the scope to include thought-provoking overviews and demonstrations of new research that crosses traditional boundaries.*
>
> - SIGGRAPH Asia Conference Committee

The first RL5 workshop was conducted at The SIGGRAPH Asia 2018 Courses program, together with Dr. Sam Green. [1] RL5 was in its early version at the time that only the Tabular Q-learning algorithm and two environments were implemented, but the fundamental framework has been used throughout the later versions.

The course was 105 minutes with the following agenda:

1. Introduction to the core concepts of reinforcement learning (10 minutes)
2. A brief survey of artworks in deep learning (5 minutes)
3. Introduction to Processing community (5 minutes)

[1] Dr. Sam Green graduated from Computer Science Department at University of California, Santa Barbara and currently works at Sandia National Laboratory as a research scientist.

4. Explanation of the Tabular Q-learning algorithm (35 minutes)

5. Implementation of the Tabular Q-learning algorithm in p5.js (40 minutes)

6. Questions & answers (5 minutes)

The first four sessions of the course were lectures and the 5th session was a hands-on practice in which I guided the participants to train an RL agent in their browsers with the provided RL5 framework. Given the novelty of the contents and the limited timeframe, the purpose of the workshop is twofold: to understand motivations of people attending the course and to evaluate the simplicity of the RL5's framework.

Roughly 60 people attended the workshop and stayed during the whole course. We conducted a qualitative survey that people can submit their responses online or verbally communicate with us right after the workshop. The two main reasons for people to attend the course are: 1) to understand reinforcement learning as a non-programmer, and 2) interested in how to use reinforcement learning and RL5 for creative applications.

According to the survey, the course was "simple, clear, and easy to comprehend." During the coding session, presenting the RL5 framework posed no difficulties.

We also received several constructive feedback regarding RL5 and the structure of the course, summarized as below:

- People who have experienced in creative coding and digital art would appreciate if RL5 could further abstract away technical details of implementing RL algorithms.

- The time allotted for the course was too short for participants to experiment with different parameters and create new RL environments.

- The course might too dense for an absolute novice.

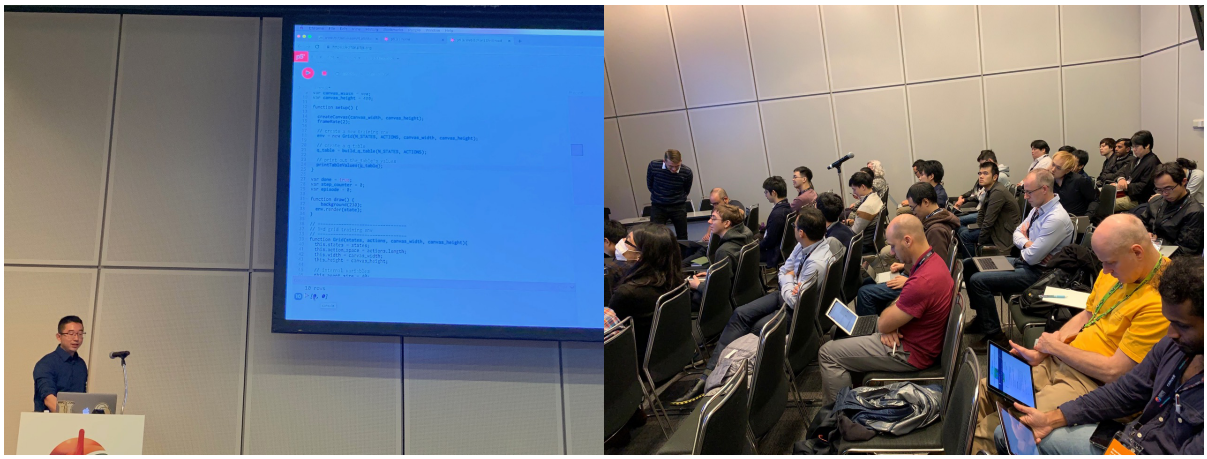


Figure 5.1: (Left) The author was introducing RL5 at the SIGGRAPH Asia 2018 Courses program. (Right) Participants of the course practiced RL5 in the live coding session.

To conclude, the course was well delivered to a group of audience with mixed backgrounds. The concept of bridging reinforcement and creativity was attractive to people who have been working in the Processing community and digital art. The fundamental framework of RL5 was simple to use for non-programmers. On the other hand, a longer workshop can help participants to explore more creative parts of RL5 and a quantitative survey could be conducted to better evaluate RL5.

5.1.2 Digital Media Design at Beijing University of Technology

The second workshop of RL5 was given at the College of Art and Design, Beijing University of Technology, Beijing, China. The university was established in 1960 and is recognized as one of the top 100 universities in China. The College of Art and Design consists of 9 programs and the students participating in the workshop were sophomores and juniors from the Digital Media Design program.

In this five-day workshop, a three-hour lecture was given in the morning and a three-hour lab session was conducted in the afternoon for the first three days. In the last two days, the students prepared for a public exhibition to present their final projects from

the workshop. The agenda of the workshop is below:

- **Lecture Day 1**: Introduction to Digital Arts - An overview of featured digital artworks from 1950s
- **Lab Day 1**: Introduction to p5.js - Get familiar with the basic syntax via multiple mini practices
- **Lecture Day 2**: Introduction to Reinforcement Learning - An overview of reinforcement learning's history, applications, concepts, and core elements
- **Lab Day 2**: Introduction to RL5 - Get familiar with the framework via a hands-on practice of training a grid-world agent with the tabular q-learning algorithm
- **Lecture Day 3**: Discussion of reinforcement learning and digital art - draw a conceptual connection between reinforcement learning and digital art via analyzing a group of the author's projects
- **Lab Day 3**: Define a RL environment based on the given 1D gridworld example
- **Day 4**: Prepare for the final projects exhibition - the final project is to present the customized RL environment before and after training
- **Day 5**: Final projects exhibition - present all the 32 participants' projects.

After the workshop, a user study was conducted to evaluate how the workshop helped the students understand reinforcement learning, and how RL5 reduced the technical barriers of reinforcement learning and inspired the students to use it for their future artistic explorations. A total of 32 participants participated in the user study. The user group consisted of 10 male users and 22 female users. The final results of each question are listed below:

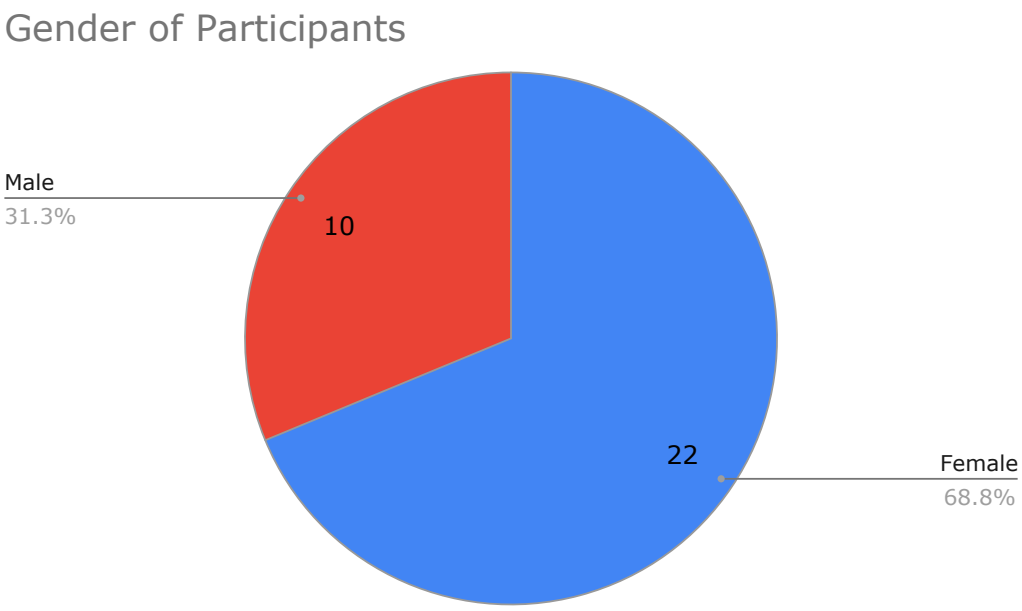


Figure 5.2: Question 1

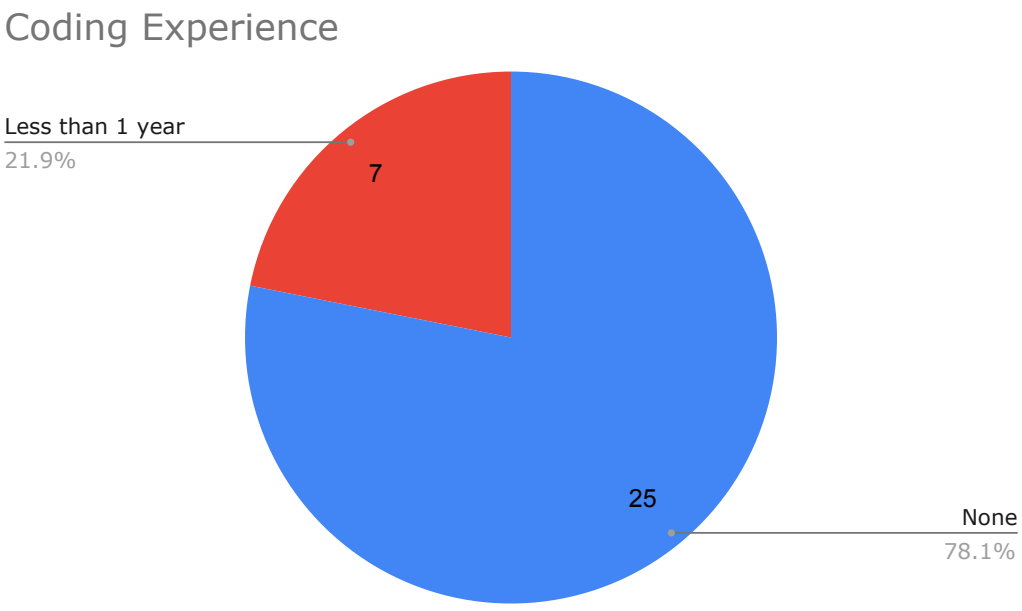


Figure 5.3: Question 2

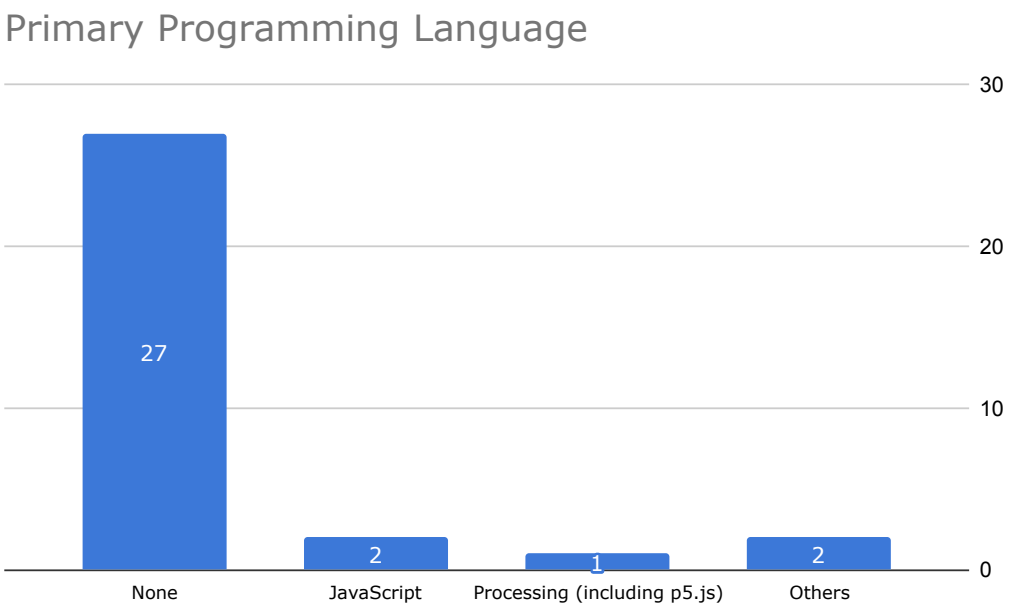


Figure 5.4: Question 3

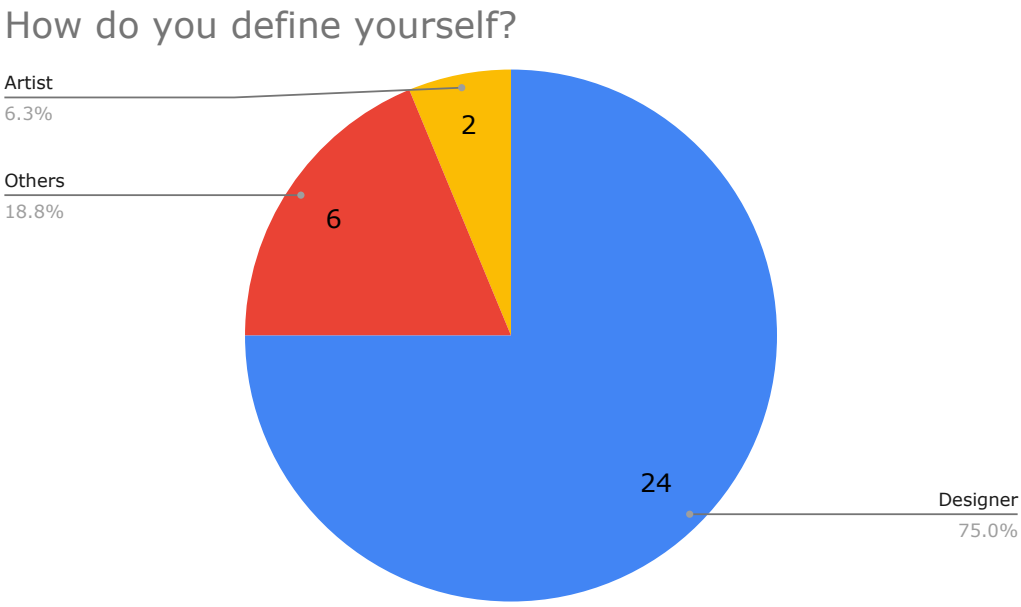


Figure 5.5: Question 4

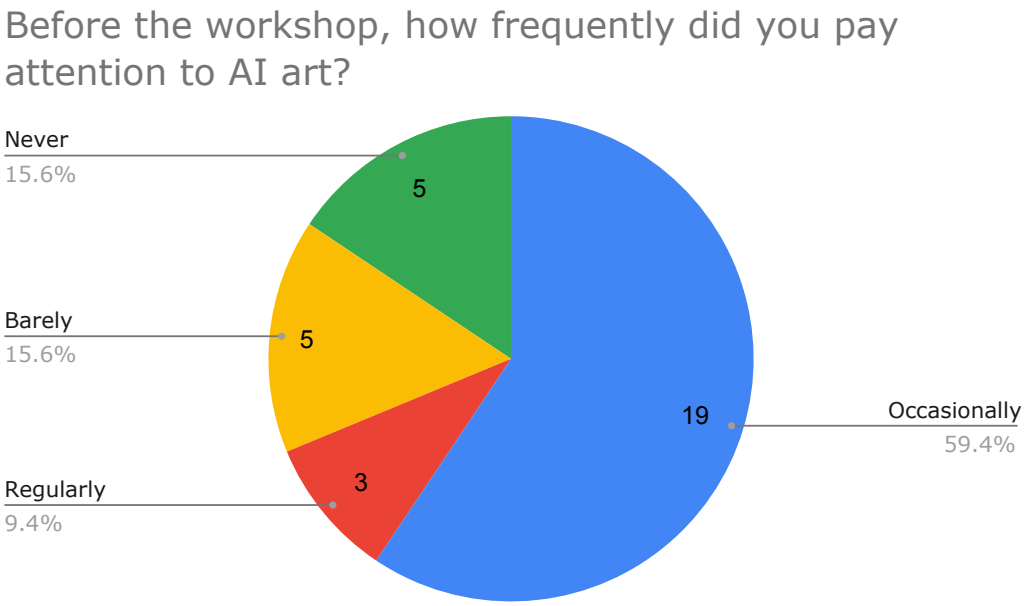


Figure 5.6: Question 5

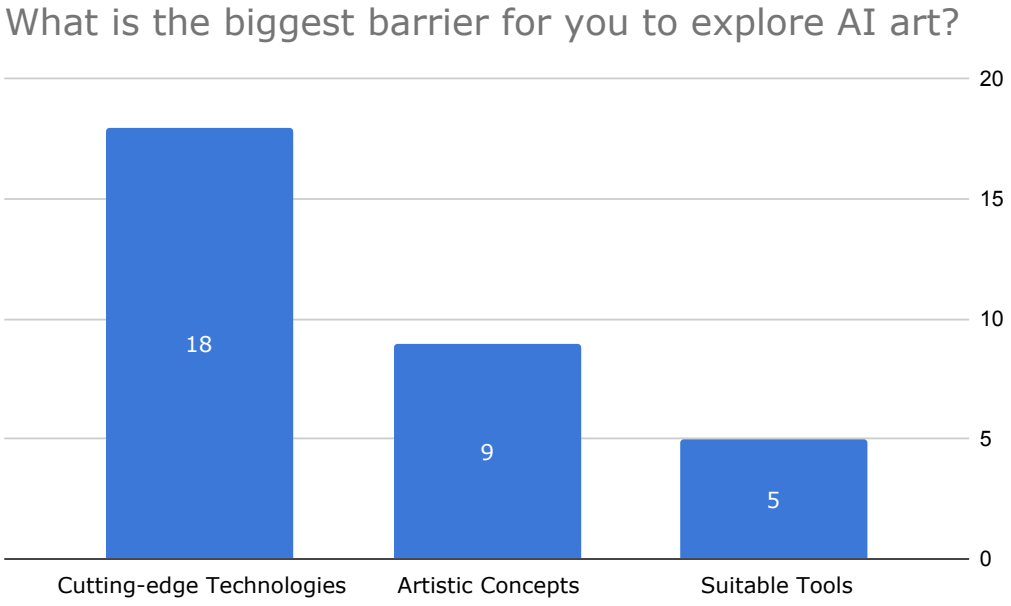


Figure 5.7: Question 6

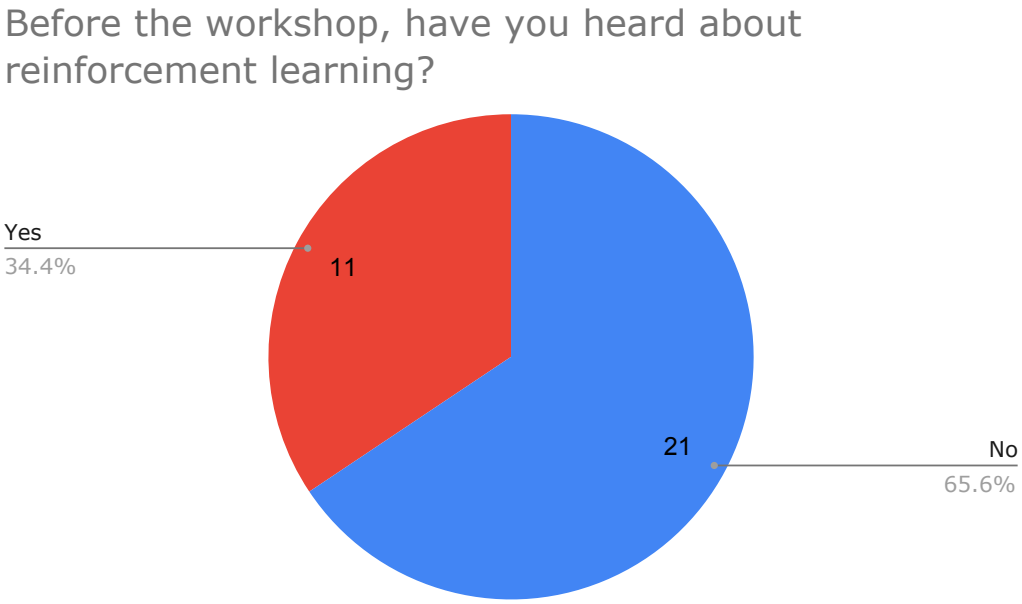


Figure 5.8: Question 7

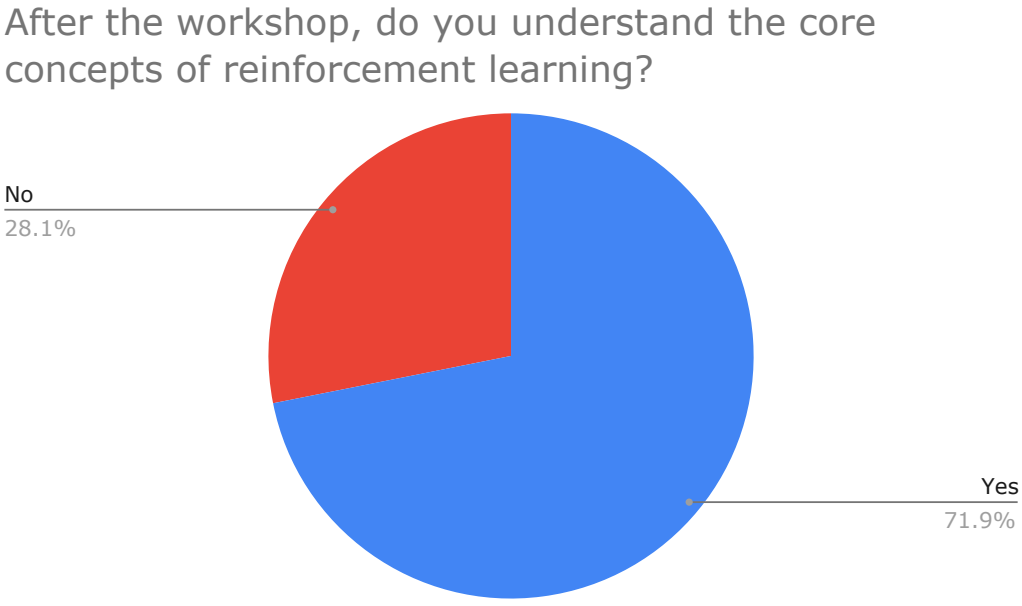


Figure 5.9: Question 8

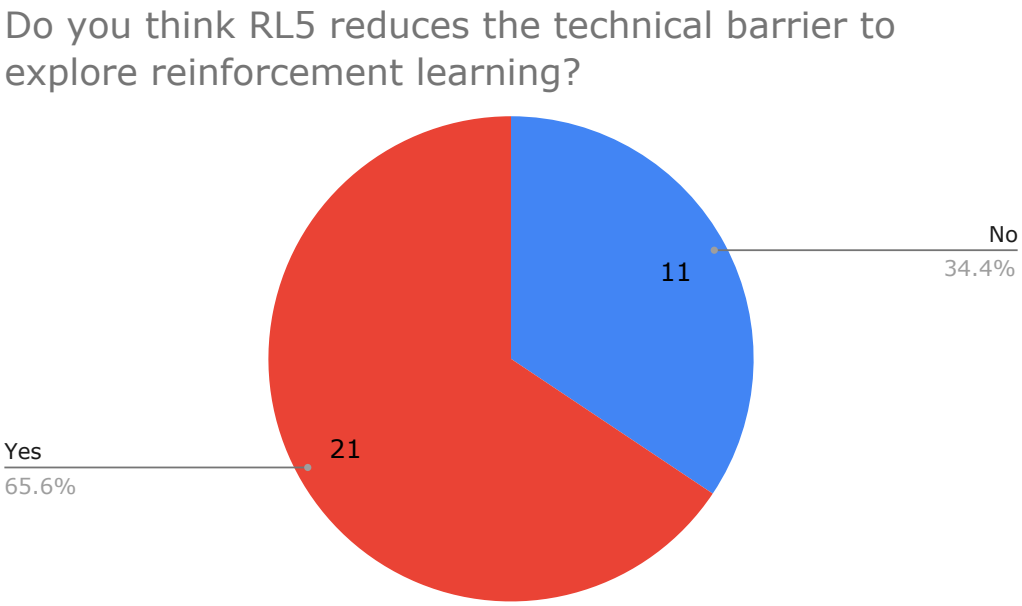


Figure 5.10: Question 9

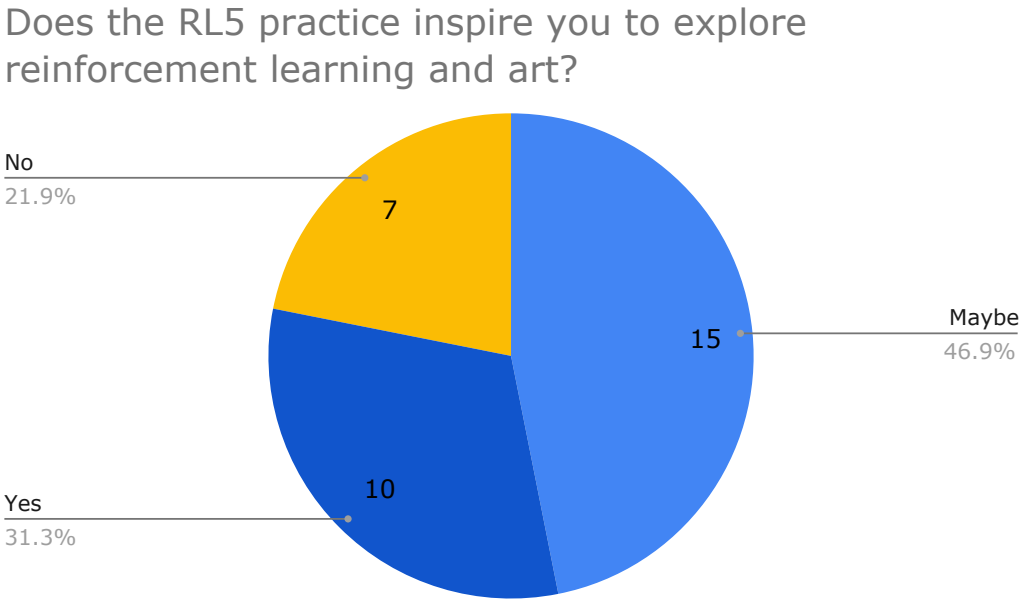


Figure 5.11: Question 10

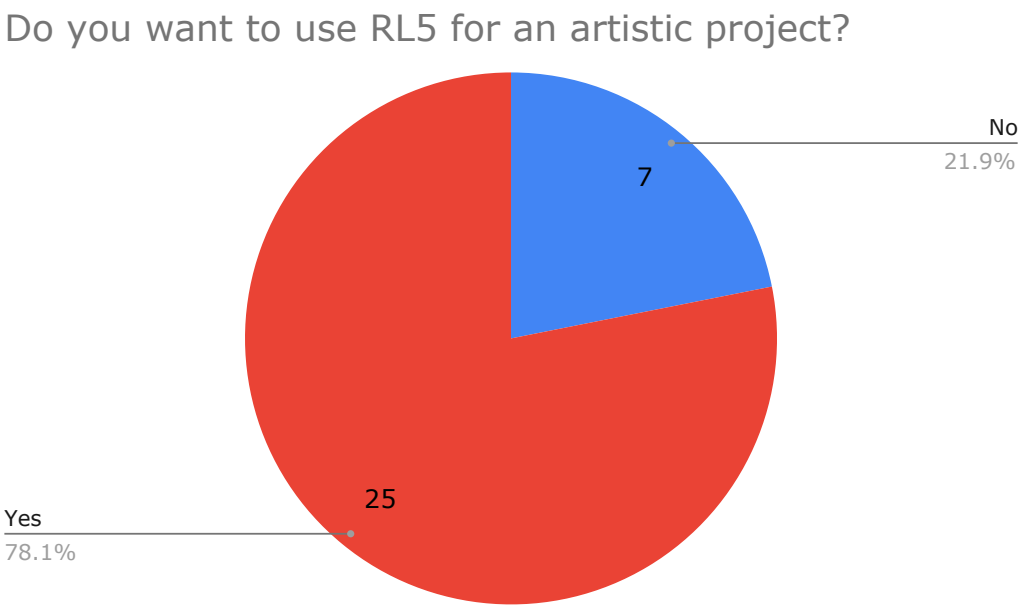


Figure 5.12: Question 11

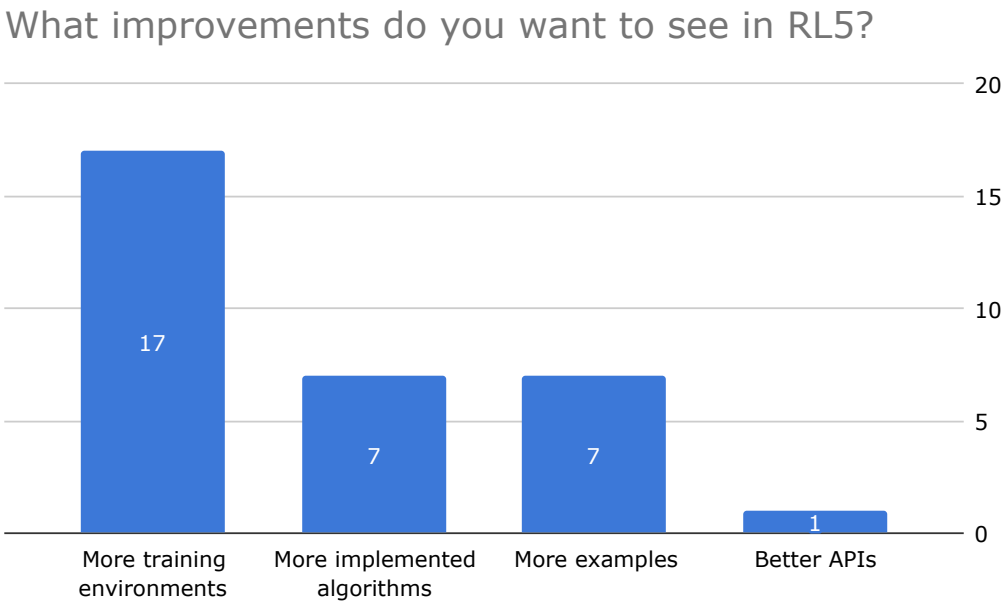


Figure 5.13: Question 12

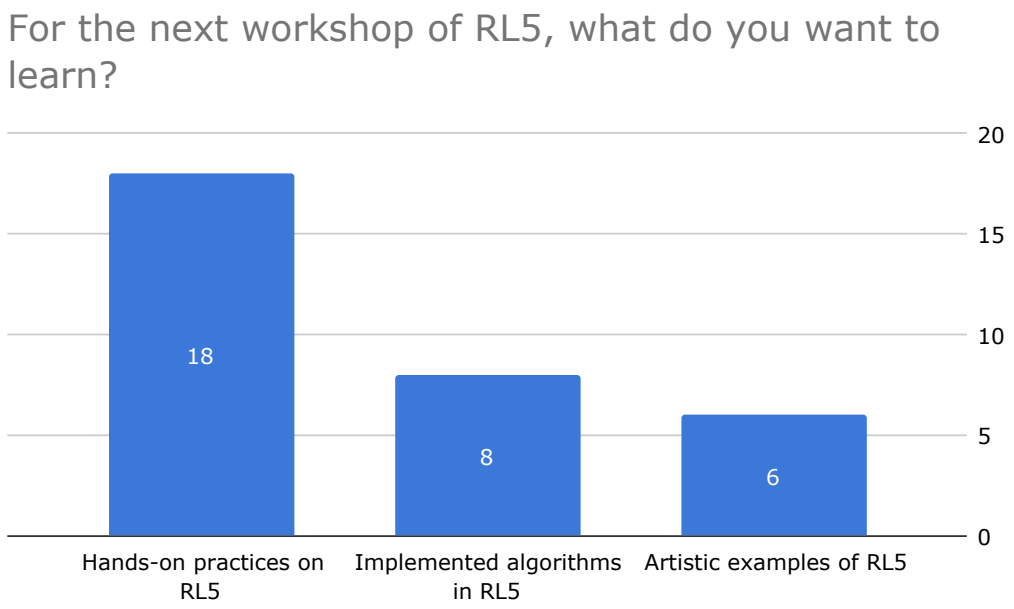


Figure 5.14: Question 13

The results of the study showed that all the participants had very limited coding experience and most of them considered themselves as a designer (Question 2, 3, and 4). Before the workshop, the majority of them heard about AI art in general (Question 5) but didn't know too much about reinforcement learning (Question 7). More than half of the participants believed the cutting-edge technologies in AI were the biggest barrier preventing them from exploring AI art (Question 6). After the workshop, approximately two-thirds of the participants understood the core concepts of reinforcement learning and believe RL5 could help them reduce the technical barrier of reinforcement learning (Question 8 and 9). Roughly three-fourths of the participants were inspired by the RL5 practice in the workshop for their future artistic exploration of reinforcement learning and would like to use RL5 for an artistic project (Question 10 and 11). Most of them were satisfied with the APIs of RL5 but would like to see more training environments in RL5 and acquire more hands-on experience of RL5 in the future (question 12 and 13).

5.2 Artistic Applications

5.2.1 *Machine Learning with Visual Overlay*

Completed year: 2019

Type: RL-based Geneartive Art

Collaborator: Andreas Schlegel [2]

Machine Training with Visual Overlay (MTwVO) is a screen-based generative art simultaneously presenting two visual programs that one program (A) is indirectly driven by another program (B). Specifically, program A is a real-time image-generating shader system performing Ray Marching, a technique for rendering complex volumes and hyper-realistic scenarios in 3D spaces. Program B trains an RL-agent to solve the 1D Gridworld task with the tabular q-learning algorithm and visualizes the footprints of the RL agent.

MTwVO works in cycles. A cycle begins with a reset of the two programs and ends when the RL-agent has been successfully trained and evaluated a few times. At the beginning of each cycle, program A renders a sphere and program B clears the table of the RL agent. As the training goes, the sphere transforms into abstract shapes and textures based on the messages received from the RL agent. At the end of each cycle, Program A saves a screenshot of the current state of the rendered image. The diagram of the whole system is shown in Fig 5.15.

MTwVO is a collaborative project with Andreas Schlegel. The collaboration started by the author showing Schlegel RL5, which inspired him to explore reinforcement learning with his artistic practice. Below is a quote from Schlegel to explain his motivation for the collaboration. Appendix B presents a write-up of his thoughts on MTwVO and further

[2] Andreas Schlegel is a Senior Lecturer at LASALLE College of the Arts Singapore where he teaches across disciplines and has lead the colleges Media Lab since 2008. Currently he is affiliated with the Faculty of Design. He attained an MA from the University of Portsmouth (United Kingdom) in 2000 and a MSc at the University of California, Santa Barbara (USA) in 2004.

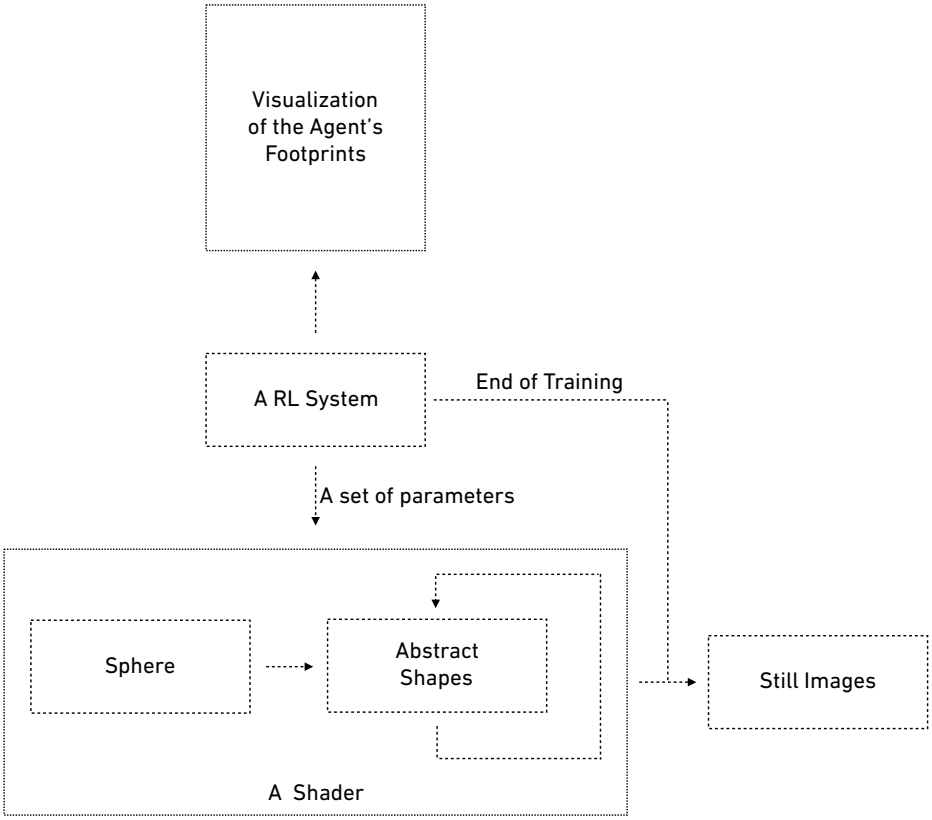


Figure 5.15: A diagram shows the whole system behind *Machine Training with Visual Overlay*

expectations of RL-based generative art.

> The initial motivation for MTwVO was the fascination that a program can learn, or at least what we can perceive as learning, to find a way to accomplish a task that is essentially a matter of trial and error and can ultimately be executed perfectly. It is important to mention here that it makes a significant difference to see this process unfold rather than reading about it in a paper. The process becomes tangible and accessible as the respective source code can be read, used and modified.

The project was premiered at 2019 Singapore Art Week, the leading visual arts festival

in Southeast Asia [3]. The installation at the festival consists of two screens attached to a stand on wheels. The smaller screen displays the RL footprints in black and white lines and the larger screen presents the constantly changing abstract shapes rendered by the shader program. Fig 5.17 shows the setup at the festival and Fig 5.16 shows a close-up of the visualization of the RL agent's footprints.

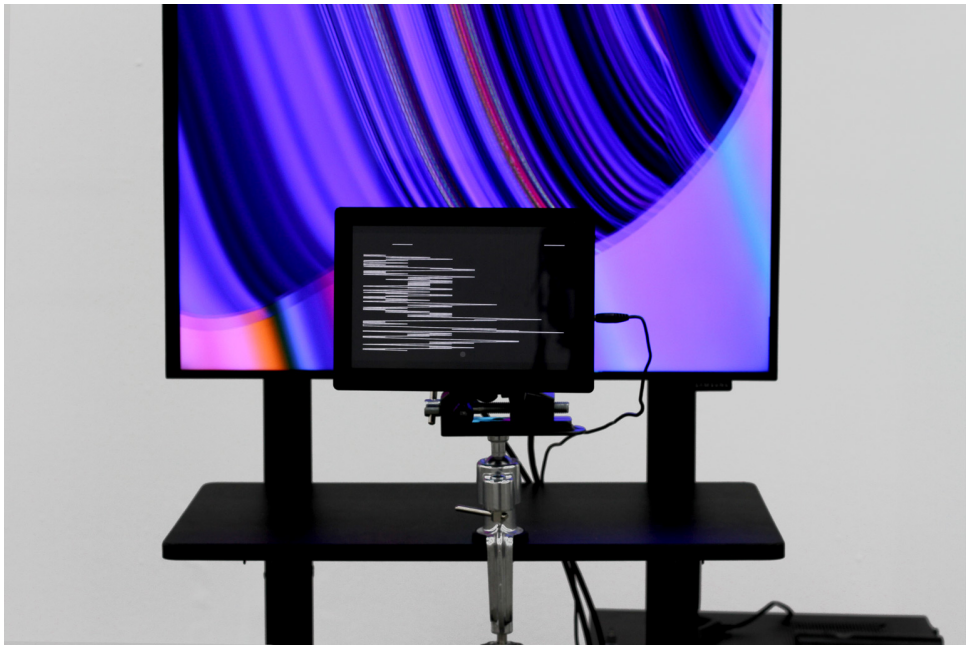


Figure 5.16: A close-up of the screen visualizes the RL agent's footprints. *Machine Training with Visual Overlay* at 2019 Singapore Art Week

5.2.2 *Action Through Inaction*

Completed year: 2019

Type: RL-based Geneartive Art

The title *Action Through Inaction* derived from a Chinese philosophical concept "wu wei", literally meaning "effortless action". The concept was originally from Confucianism

[3]https://www.artweek.sg/

Figure 5.17: The installation of *Machine Training with Visual Overlay* at 2019 Singapore Art Week

and became an important concept in Taoism. It denotes a state of mind of taking effortless and spontaneous actions to acquire an almost magical efficaciousness in moving through the world.

In *Action Through Inaction*, a constraint environment exists to host an RL agent and a dangerous zone to the agent. The goal of the agent is to develop a strategy to avoid the dangerous zone while move freely in the environment. When the agent achieves the goal, the environment resets by randomly placing the dangerous zone to a new location. The agent bounces back when it hits a boundary of the environment. Visually, the environment is a grey 2D rectangular area with a white border. The dangerous zone is a light grey rectangular area within the environment. The RL agent is described as a red dot and its footprints are visualized as a sequence of white dots. Fig 5.18 shows a screenshot of the scene rendered in p5.js.

Regardless of the location of the dangerous area, the agent always develops a strategy of performing parabola-like movements in a corner away from the dangerous area. Fig 5.19 shows one strategy developed by the agent. Ultimately, the agent may develop a strategy to stop moving and stay at a spot until the environment resets.

The project started as a response to a current perspective on artificial intelligence (AI) that AI will eventually replace human beings as it develops and evolves. The fear has been introduced in many scenarios involving the developments of robots. Boston Dynamics[4], for example, an American robotic company based in Waltham, Massachusetts, is best known for the development of SPOT, a versatile quadruped robot with animal-like behaviors. According to the official announcement, the robot can successfully move from spot A to spot B in an unknown environment, including some extreme circumstances. The product has been envisioned as a deadly weapon in some movies or short films, like

[4] `https://www.bostondynamics.com/`

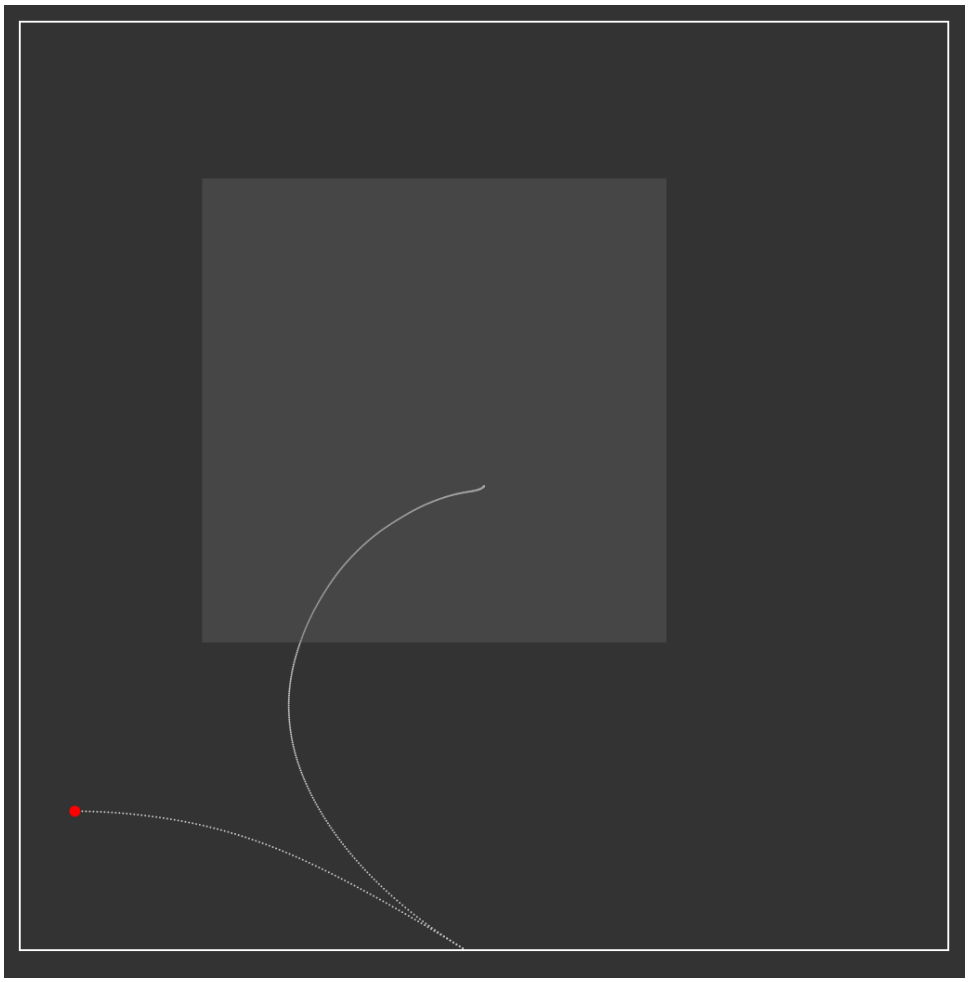


Figure 5.18: A screenshot of the scene in *Action Through Inaction*.

Metalhead in *Black Mirror* [5].

However, in reinforcement learning, which has been heavily applied to intelligent robotics, RL agents usually tend to use minimal efforts to achieve the assigned tasks. In *Action Through Inaction*, the agent starts by massively exploring the space, which is in its chaotic status. After a few interactions of exploration, it can always avoid the dangerous zone by taking less efforts, which enters its adaptive status. Fig 5.20 shows the changes in the behaviors in one training session. The training process took approximately 10 minutes.

[5] `https://en.wikipedia.org/wiki/Metalhead_(Black_Mirror)`

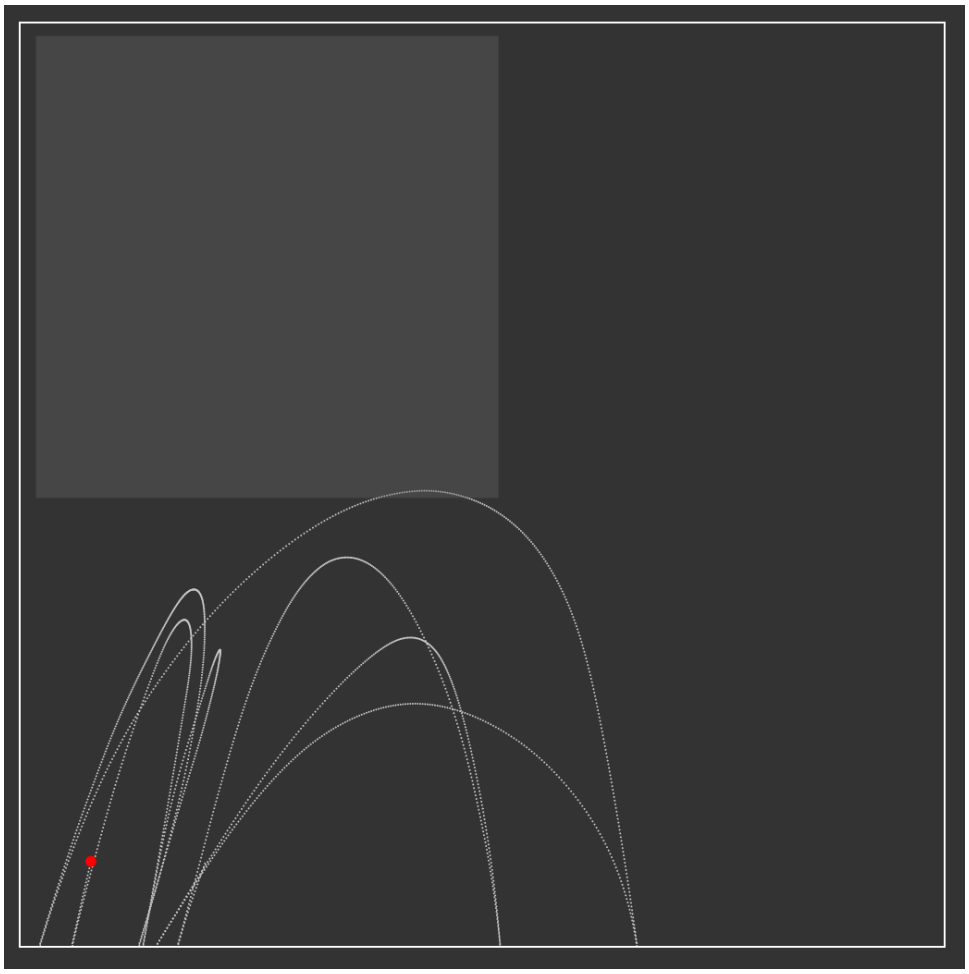


Figure 5.19: One strategy developed by the agent to stay away from the dangerous area. The white curves indicate the footprints of the agent in the last 2000 steps.

Aesthetically, the project was inspired by traditional Chinese landscape painting, where mountains and waters are usually the main subjects in this art form. *Action Through Inaction* further abstracts away the details of mountains and waters by using circles and rectangles. The red RL agent functions as the moving focal point in the painting. The artwork can be presented as a never-ending animation [6] or still images composed by selected screenshots from the scene, like the one in Fig 5.21.

[6] A recording of the animation can be seen at `https://vimeo.com/384725054`

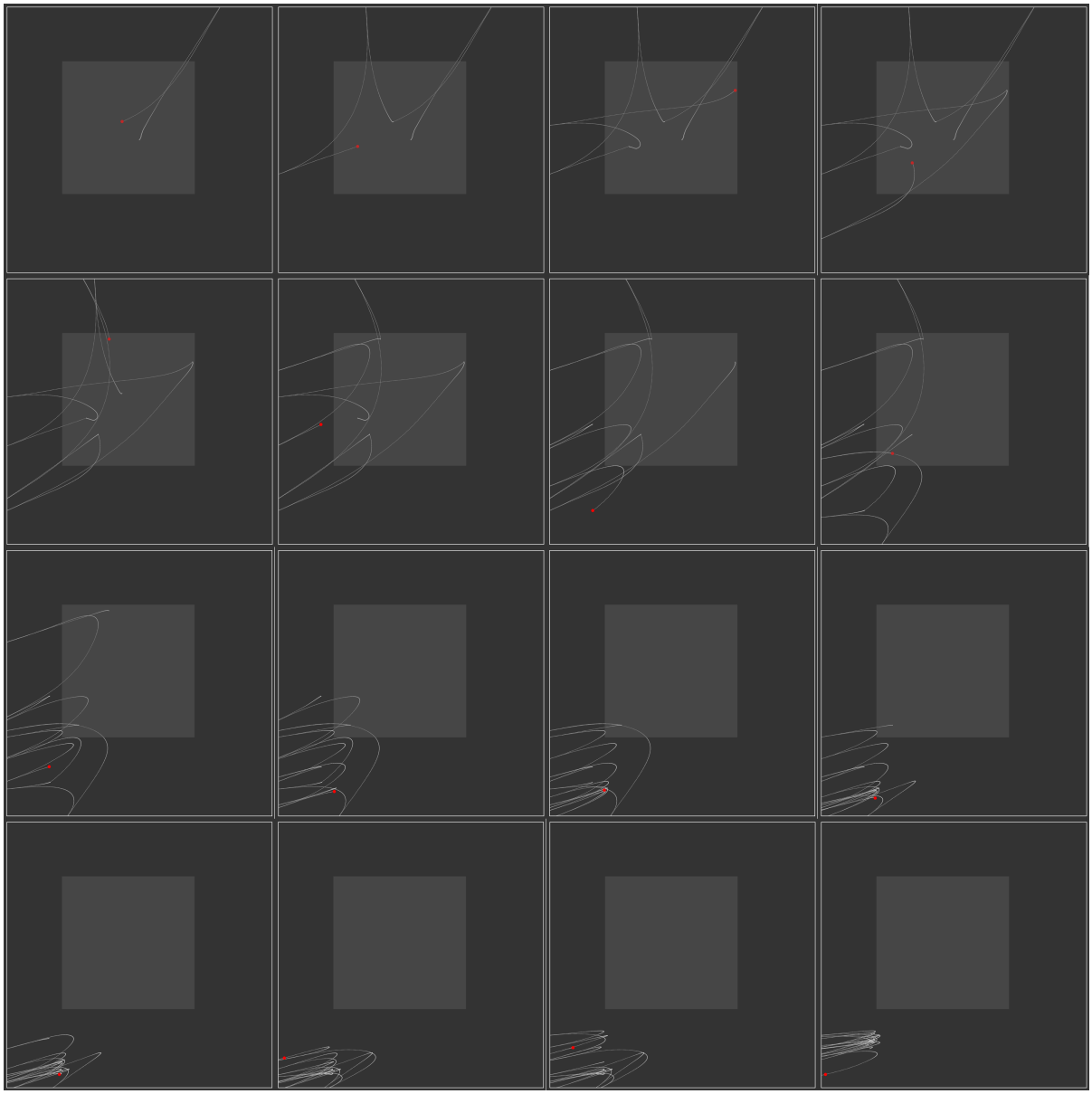


Figure 5.20: An RL agent's behavior changes over the training in *Action Through Inaction*. From left to right and top to bottom, each grid presents the agent in every 500 steps, starting from step 0. The path of the footprints includes the most recent 2000 steps.

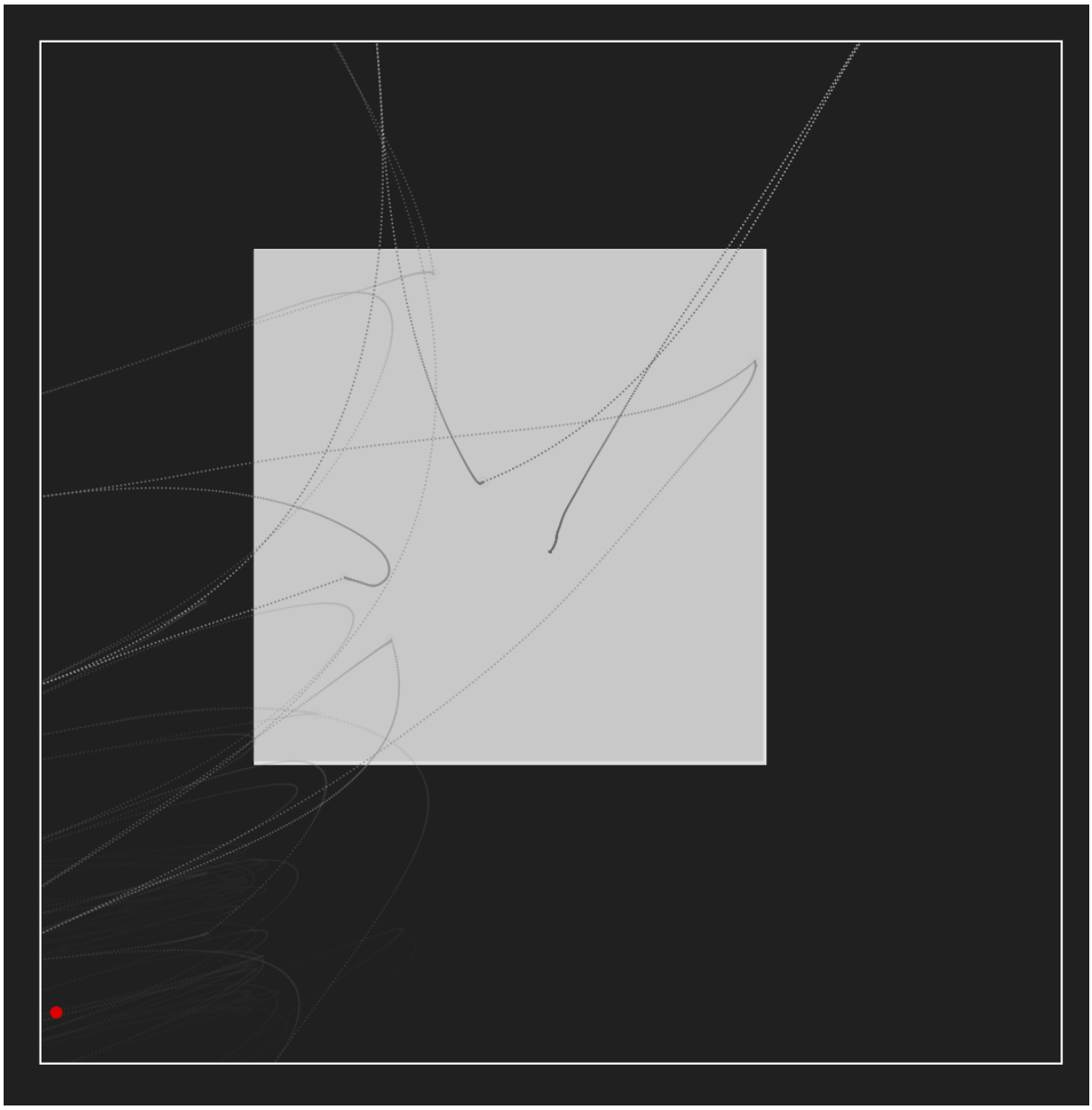


Figure 5.21: A still image derived from *Action Through Inaction*, representing the lifetime of a RL agent and its interactions with the dangerous zone.

Chapter 6

Review & Discussion

In this dissertation we have investigated the potential of reinforcement learning for artistic applications. Chapter 1 covered the motivation, problem statement, challenges, and approaches for the dissertation. Chapter 2 provided a high-level introduction of reinforcement learning, its applications in games and robotics, and the state-of-art RL libraries. The chapter also reviewed generative art and deep-learning art by definitions and examples. Chapter 3 introduced a framework adapting the evolving process of RL systems for artistic explorations and discussed related achievements in generative art, deep-learning art, and reinforcement learning. Chapter 4 described RL5, a JavaScript library built on top of p5.js for rapidly prototyping RL environments and training RL policies in web browsers. Chapter 5 demonstrated four use cases of RL5, including two workshops of RL5 and two artistic applications of RL5.

Reinforcement learning is an efficient class of sequential decision-making algorithms aiming at training an agent to maximize the total reward it receives in the long run from an environment. At each time step t, the agent is given information about the state s_t of its environment and then makes an action a_t. The agent's policy $\pi(s_t)$ is the logic that takes observations and returns actions. After each action, the environment

will return a new state s_{t+1} and reward r_{t+1}. As a computational approach focusing on goal-orientated learning through interaction, reinforcement learning is distinguished from supervised learning and unsupervised learning by emphasising the learning process through direct interaction with the environment, without requiring prepared training data. The emphasis of learning through interaction is one of the key characteristics of reinforcement learning.

Since the initial purpose of reinforcement learning is to train an agent to solve a specific task in an uncertain environment, RL problems are usually task-oriented. For example, in *Dynamic Experience Replay* [84], the goal was to learn an optimal policy, which takes Cartesian pose and force/torque observations as input and outputs Cartesian velocity for high-precision assembly tasks. However, this task-oriented approach limits applying reinforcement learning in generative art. As a rule-based art form, a common goal of generative art is producing unpredictable behaviors driven by the defined rules. This goal is hard to be measured as achieved or failed, which raise the difficulty to define the reward function in reinforcement learning. Besides the steep learning curving, this factor also contributes to the situation that only a few RL-based generative art projects exist in the literature. In the NeurIPS Workshop on Machine Learning for Creativity and Design 3.0 [1], only two [85, 86] out of thirty-one projects directly addressed reinforcement learning.

This dissertation suggests a process-oriented method to examine RL-based generative art. Instead of focusing on training a creative agent, the method considered the entire training process of an RL agent as an *unpredictable evolving process*. In an RL system, the agent might develop peculiar solutions for a simple problem, or it might disobey the creator's intention and develop obscure behaviors. While current RL research primarily focuses on how to train and deploy RL policies in the real world more efficiently and

[1] https://neurips2019creativity.github.io/

accurately, examining the unpredictability in this evolving process can bring important insights to the philosophical question of how the machine perceives the world. Artists can play an essential role in deliberately designing RL environments to highlight such counter-intuitive behaviors developed by the RL agents and use those ever-changing behaviors to explore novel aesthetics.

It is important to acknowledge that the training processes in reinforcement learning share certain similarities with the processes in genetic algorithms where both methods start with random explorations and end with an optimal solution. However, conceptually, the two methods are fundamentally different. Reinforcement learning usually uses one agent to perform the trial-and-error search, while a genetic algorithm typically starts from a population of agents and iteratively selects the better fitted agents. It is worthwhile to compare the behaviors of the two methods (reinforcement learning and genetic algorithms) in diverse environments in the future.

RL5 was developed with the purpose for non-experts to practice and explore reinforcement learning. With the particular focus on simplicity, the library enables developers to train and evaluate an RL agent within 35 lines of codes. Besides, developers can define their RL environments in native p5.js syntax with a provided structure. The intention of developing RL5 is twofold: to serve as a pedagogical platform for non-experts to practice reinforcement learning, and to serve as a creative platform for users to rapidly test ideas and prototype RL projects. The library was implemented in two workshops and two artistic applications.

During the two RL5 workshops, users had no difficulties in training and evaluating RL agents with the provided examples on their laptops. In each workshop, the participants were able to observer the RL training process within 10 minutes. I found that observing the real-time training process is an essential part for the participants to develop an intuitive understanding of the trial-and-error search process of reinforcement learning, as

many of them expressed confusion when my collaborator or I introduced the concepts of reinforcement learning in mathematical formulas. The students' final projects from the second workshop also indicated that they understood the core concepts of reinforcement learning.

The simple examples in RL5 also facilitate users to experiment with hyper-parameters in RL algorithms. Hyper-parameters like *learning rate* [2] or *epsilon greedy* [3] can directly affect the training results, but the process of understanding the hyper-parameters requires expertise and experience. Those simple examples enable users to rapidly conduct ablation studies on the hyper-parameters in the RL algorithms in order to develop a deeper understanding. Further, the experiments could be inspirational for artistic explorations. For example, one participant observed that assigning different values for epsilon greedy can result in different characters of an RL agent.

The results of the survey conducted after the second workshop indicated that non-programmers could prototype simple RL environments with some basic training in p5.js. It suggested that RL5 can not only serve as a platform for non-experts to study reinforcement learning, but also can be extended as a platform for teaching creative programming for non-programmers.

One limitation of RL5 revealed from the workshops is the variety of training environments. The environments of the steering behaviors provide a promising direction for the students to build a complex system using behaviors like seek, avoid, or follow. On the other hand, it also limits the students' creativity as they did not explore other possible scenarios. This phenomenon could also be contributed by other aspects, like the time-frame of the workshop or the coding experience of the participants. Most of the students had minimal coding backgrounds, and they only had one day to work on the project. A

[2] *Learning rate* refers to how fast an RL agent can learn. The range is between 0 and 1.

[3] *Epsilon greedy* refers to the probability for an RL agent to execute the current policy versus take random actions.

10-week course with a variety of students could better examine the pedagogical part of the library.

The two artistic applications using RL5 investigated the process-oriented method of RL-based generative art. In *Machine Learning with Visual Overlay*, the artwork visualized the training process of an RL agent on a small screen with the goal to move from the leftmost state to the rightmost state in a 1D gridworld. The visualization was two dimensional where the vertical axis represented time, and the horizontal axis represented the current location of the RL agent. As the training proceeded, the patterns of the visualization transformed from zig-zag shapes to straight lines. Figure 6.1 shows a complete training session in the visualization. The second part of the project is a shader program using ray marching to generate 3D abstract shapes continually changing over time. The shader program contains ten parameters responding to the ten states of the RL agent. At each timestep, the current state of the agent triggered the corresponded parameter in the shader to generate a random value to influence the 3D shape.

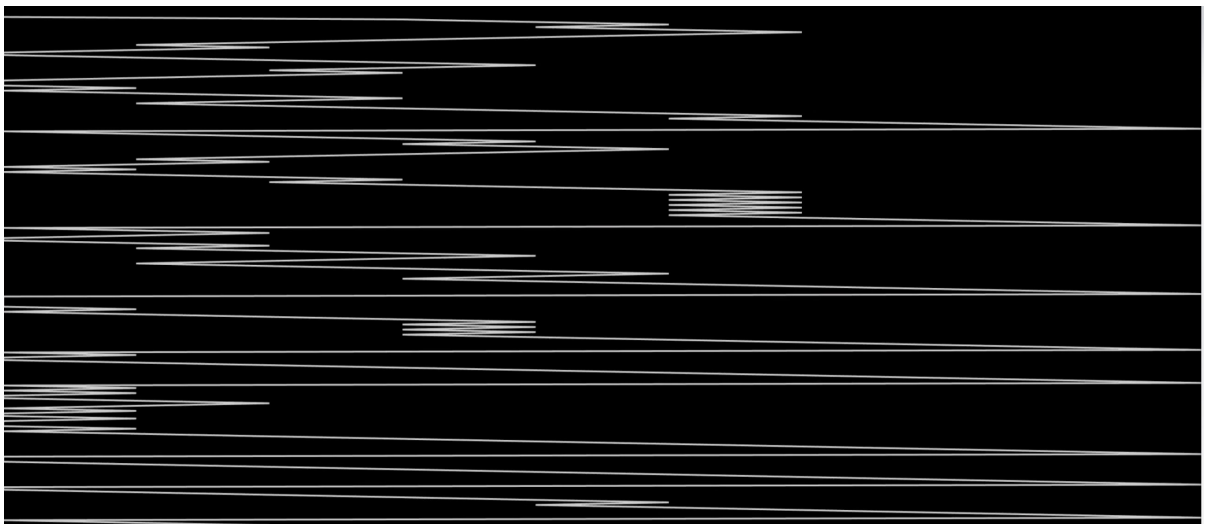


Figure 6.1: This visualization shows the performance behavior of the RL agent's training process. In between the start and the bottom completion, the visualization shows how the process goes through incomplete and hesitant, repeating states until achieving the optimal behavior as seen at the bottom where there is continuous movement to the right.

The artists' intention behind *Machine Learning with Visual Overlay* was to explore how a simple RL task could be leveraged for complex visual systems. As reinforcement learning is a technique for sequential decision-making problems, it's not feasible to directly output a complete image as demonstrated by GANs or DeepDream. Attempts [56, 87] have been made to use RL agents as brush strokes by receiving rewards from a trained discriminator. In *Machine Learning with Visual Overlay*, we showed an RL agent could also be used as a controller for a sophisticated visual program where the deforming pattern can indirectly reflect the random-to-order process. One limitation of presenting the evolving process of an RL system in such a complex visual system is the difficulty of distinguishing the process with a random process. As the creators of the project, we could recognize if the RL agent was trained by observing the visual patterns in the shader program, but we realized that it was difficult for audiences to appreciate the value of reinforcement learning by only watching the patterns in the shader program. One solution we provided was to set up another screen to visualize the training process of the RL agent directly. However, it was still unclear for audiences who hadn't approached reinforcement learning before.

Inspired by the project, I found that an RL agent could be applied as a selective agent in evolutionary art where the artist or audiences are usually the selective agent. The system behind *Machine Learning with Visual Overlay* was a feed-forward system where the RL agent didn't receive any messages from the shader program. Hence, the RL agent is not selective for the shader program as the agent has no knowledge to the program. If the system were extended to a closed-loop system where the RL agent could take the feedback of the shader program, part or in whole as reward signals, the agent could develop a better understanding of the program in order to be selective. The details of the forms of feedback and reward function need to be further investigated. On the artist side, Schlegel was more familiar with programming paradigms like imperative,

procedural, or object-oriented programming. Working with RL5 in *Machine Learning with Visual Overlay* allowed him to experiment, play, and develop an understanding of reinforcement learning that he could apply in his future artistic practice.

In comparison to *Machine Learning with Visual Overlay*, which explored the RL training process in a practical way, *Action Through Inaction* investigated a philosophical meaning of the training process. The intention of *Action Through Inaction* was to discuss what intelligence means in the context of artificial intelligence. In reinforcement learning, an agent can be adaptive to solve problems in uncertain environments. That is the intelligent part in an RL agent. However, given the current techniques, it is unlikely that an RL agent would develop extra skills to solve new problems. I found it was exciting to use simple RL environments to represent abstract philosophical ideas, like effortless action indicated in *Action Through Inaction*. The project also showed a possibility to create novel aesthetics based on the behaviors of RL agents. *Action Through Inaction* demonstrated a basic usage of the RL agent's footprints by simply visualizing the latest 500 steps as white dots on their locations. Since each step contains multi-dimensional data, like position, time, or reward, the system behind *Action Through Inaction* has the great potential to be extended into more complex visual effects. Another phenomenon I noticed was that some audiences developed an emotional attachment to the agent. They expressed pity when the agent entered the dangerous zone. I suspect the emotion was caused by expectations and unpredictability carried out during the training process. An important line of work will be to quantify the observation via controlled human studies.

One shortcoming of *Action Through Inaction* was that the simple environment might suggest reinforcement learning is limited in artistic explorations. It is worth noting that another purpose of *Action Through Inaction* is to demonstrate RL5 with the aim to train an RL agent in a few minutes with the real-time rendering. The framework of *Action Through Inaction* is expandable for more complex environments, like the dangerous

zone could move, multiple dangerous zones exist with different penalties, or the zone has two states that one provides penalty and anther one provides reward. Each environment would result in different behaviors of the RL agent, which opens up numerous opportunities to be explored.

Chapter 7

Conclusion

7.1 Contributions

This dissertation explores on adapting reinforcement learning for generative art. The main contributions include a conceptual framework of adapting reinforcement learning to generative art, a JavaScript library called RL5 to improve the accessibility of reinforcement learning for non-experts, and a group of pedagogical and artistic applications of RL5.

The dissertation proposes a term *RL-based Generative Art* to denote a novel form of generative art in which the use of RL-agents is the key element. To explore RL-based generative art, the dissertation discusses a creative process in RL-based generative art that artists need to be engaged with RL environments by defining visuals, RL-agents, reward functions, and transition dynamics. In addition, artists need to decide how to present an RL system as an artwork. The dissertation then introduces three aspects to result in emergent behaviors in RL-based generative art, which are random exploration, state & action spaces, and reward function. An RL environment is created to carry out several experiments on emergent behaviors. Several authors related practices on

generative art, deep learning art, and reinforcement learning have been introduced. Those practices are critical for the author to understand the conceptual and technical details of each component in order to construct the framework. All the practices have been either exhibited or published at major conferences or gallerias.

To better engage with RL-based generative art, the dissertation creates RL5, a JavaScript library built on top of p5.js to improve the accessibility of reinforcement learning for creatives. RL5 allows developers to define their own RL environments in native p5.js language and train RL policies in web browsers. RL5 provides three RL algorithms to cover the four possible combinations of different types of state and action spaces, and nine RL environments to serve as building blocks for constructing complex systems. With the focus on simplicity and (re)usability, the APIs of RL5 enables users to create, train, and evaluate an RL agent in less than 20 lines of codes. The library is demonstrated in an RL environment called *Avoid An Obstacle* in which the goal is to train an agent to move on a 2D rectangular area from left to right without hitting a rectangle in the middle. With the same training settings but different random seeds, the agent develops different strategies to accomplish the task.

The dissertation has further evaluated and practiced RL5 on two pedagogical applications and two artistic applications. The first pedagogical application was a workshop conducted in the 2018 SIGGRAPH Asia Courses program, which is a selective program aiming to push the boundaries of computer graphics and interactive techniques. In the 105-minute course, the framework of RL5 was well delivered to approximately 60 audiences with mixed backgrounds. According to a qualitative survey after the workshop, the participants believed the workshop was simple, clear, and easy to comprehend, and the concept of adapting reinforcement learning for generative art was intriguing to creatives.

The second pedagogical application was a five-day workshop given at the Digital Media Program at Beijing University of Technology. 32 undergraduates participated in

the workshop and created their own RL environments by the end of the workshop. A quantitative survey was conducted after the workshop. It showed that approximately two-thirds of the participants understood the core concepts of reinforcement learning and believe RL5 could help them reduce the technical barrier of reinforcement learning after the workshop. Roughly three-fourths of the participants were inspired by the RL5 practice in the workshop for their future artistic exploration of reinforcement learning and would like to use RL5 for an artistic project. Most of them were satisfied with the APIs of RL5, but would like to see more training environments in RL5 and acquire more hands-on experience of RL5 in the future.

The two artistic applications used RL5 to explore different aspects of RL-based generative art. *Machine Learning with Visual Overlay* explores the training process of an RL agent by visualizing the evolving history of the agent and using it as a controller for a shader program. It was a collaborative project with Schlegel, who designed the shader program. During the collaboration, RL5 showed its flexibility to integrate with advanced visual programs. The project was premiered at 2019 Singapore Art Week, the leading visual arts festival in Southeast Asia.

The second artistic application *Action Through Inaction* uses an RL-agent to depict an ancient Chinese philosophical concept about effortless action, which serves as a conceptual response to a current perspective of artificial intelligence that AI will eventually take over human beings. In *Action Through Inaction*, the agent always intends to develop a strategy to accomplish the task with minimal efforts, regardless of how the environment changes. The project has been submitted to the SIGGRAPH 2020 Art Gallery.

7.2 Future Directions

In the development of RL5 and from the surveys of the workshops, there was a need to create more RL environments. Since reinforcement learning has made great success in robotics, an interesting direction could be adding a robotic module in RL5. The module should include a frictionless 2 degree-of-freedom robotic arm and several basic tasks, like push or move. Similar to the steering behaviors, the RL environments in the robotic module should serve as building blocks to develop RL-based generative art in robotics. A further vision of RL5 would be establishing an online gallery that developers can share their designed RL environments.

Regarding RL-based generative art, a direction to explore is a system hosting multiple RL agents. An interesting thought experiment would be having three agents in an environment that the goal of each agent is to chase one and avoid the other one. The agents can be trained individually and be evaluated in a group. In the scenario, the behaviors of the agents are hard to predict. The number of agents could be scaled up to thousands and the goal of each agent could be different. That could lead to an exploration of swarm behavior in RL-based generative art.

Another possibility is to apply RL-based generative art to a physical environment in which the agents can sense their environment and express themselves through sounds, lights, or movements. The agents are curious about each other but they dont understand each other nor their surroundings. It would be interesting to explore if the agents could develop a common language in order to communicate with each other through reinforcement learning.

To conclude, the development of RL5 and the exploration of RL-based generative art will be parallel and reciprocal. The advanced RL techniques will be incorporated in RL5 to technically facilitate the development of RL-based generative art. The practices of

RL-based generative art will inspire RL5 for its future iterations.

Appendix A

Dynamic Experience Replay

Jieliang Luo[1], Hui Li

Abstract: We present a novel technique called Dynamic Experience Replay (DER) that allows Reinforcement Learning (RL) algorithms to use experience replay samples not only from human demonstrations but also successful transitions generated by RL agents during training and therefore improve training efficiency. It can be combined with an arbitrary off-policy RL algorithm, such as DDPG [24] or DQN [30], and their distributed versions.

We build upon Ape-X DDPG [72] and demonstrate our approach on robotic tight-fitting joint assembly tasks, based on force/torque and Cartesian pose observations. In particular, we run experiments on two different tasks: peg-in-hole and lap-joint. In each case, we compare different replay buffer structures and how DER affects them. Our ablation studies show that Dynamic Experience Replay is a crucial ingredient that either largely shortens the training time in these challenging environments or solves the tasks that the vanilla

[1]This paper was finished during the author's internship at Autodesk Research in 2019. The paper was initially presented at 2019 Conference on Robot Learning, Osaka, Japan.

Ape-X DDPG cannot solve. We also show that our policies learned purely in simulation can be deployed successfully on the real robot. The video presenting our experiments is available at `https://sites.google.com/site/dynamicexperiencereplay`

1 Introduction

Industrial robots have been heavily used in manufacturing and other industries, however, as they rely on pre-defined trajectories, they require precise calibration and fail to adapt to uncertainties. Adaptability to imprecision, varying conditions, and less structured environments is key to the future of automation. Reinforcement Learning (RL) has recently led to a range of successes in solving sequential decision-making problems, including learning control policies for robotic tasks. The control policies are learned through agents interacting with their surrounding environments and hold promises for generalizing to new scenarios in reaction to real-time observations [26, 88].

We focus on robotic assembly tasks that involve contact forces, because such tasks are widespread in industrial applications and yet challenging for robots to do. When the assembly pieces are in contact with one another, pose observations (direct from motion capture or indirect from perception learning models) alone are often insufficient. We explicitly consider force/torque observations for policy learning. During training, we randomize the initial condition within a pre-defined range and show flexibility of the learned policy to varying conditions.

Most of the recent success in RL was achieved using model-free methods [25, 30, 24, 89, 17]. They tend to achieve optimal performance, are generally applicable, and are easy to implement, but it is achieved at the cost of being data intensive. Leveraging human demonstrations [33] as well as various experience replay [29, 31, 32] has shown to

improve data efficiency.

We present a novel technique called Dynamic Experience Replay (DER) that allows RL algorithms to use experience replay samples not only from human demonstrations but also successful transitions generated by RL agents during training and therefore improve training efficiency. It can be combined with an arbitrary off-policy RL algorithm, such as DDPG or DQN, and their distributed versions. DER can be seen as a technique of over-sampling the under-represented class (successful trajectories in our case) from an imbalanced dataset, which has been studied and addressed in supervised learning [90, 91].

We build upon Ape-X DDPG and demonstrate our approach on robotic tight-fitting joint assembly tasks, in particular, peg-in-hole and lap-joint tasks. In each case, we compare different replay buffer structures and how DER affects them. Our ablation studies show that Dynamic Experience Replay is a crucial ingredient that largely shortens the training time in these challenging environments or solves the tasks that the vanilla Ape-X DDPG cannot solve. We also show that our policies learned purely in simulation can be deployed successfully on an industrial robotic arm performing the physical tasks.

The remainder of this paper is structured as follows. The problem statement and related work are stated in Sec. 2, followed by a detailed explanation of the proposed Dynamic Experience Replay in Sec. 3. Experiment setup, results, and deployment on a real robot are presented in Sec. 4. Sec. 5 concludes the paper and proposes future work.

2 Problem Statement and Related Work

2.1 Problem Statement

The RL problem at hand can be described as learn an optimal policy $\pi_\theta(a_t|s_t)$ for choosing an action a_t given the current observation s_t in order to minimize the expected

total loss:

$$\min_{\pi_\theta} \mathbb{E}_{\tau \sim \pi_\theta}(l(\tau))$$

where θ is the parameterization of policy π, trajectory $\tau = \{s_0, a_0, s_1, a_1, ..., s_T, a_T\}$, $\pi_\theta(\tau) = p(s_0) \prod_1^T p(s_t|s_{t-1}, a_{t-1}) \pi_\theta(a_t|s_t)$, and l is the loss function of the trajectory τ.

The above equation can be solved if a dynamics model $p(x_t|x_{t-1}, a_{t-1})$ is provided, however, the dynamics model in contact-rich tasks is difficult to obtain. Alternatively, the equation can be solved by model-free RL algorithms to avoid using dynamics. DDPG is a model-free off-policy RL algorithm for continuous action spaces. In DDPG, an actor policy $\pi : S \rightarrow A$ is created to explore the space and store the collected transition (s_j, a_j, s_{j+1}, r_j) in a replay buffer R. Meanwhile, a critic policy $Q : S \times A \rightarrow \mathbb{R}$ is created to approximate the actor's action-value function Q^π.

We would like to learn an optimal policy, which takes Cartesian pose and force/torque observations as input and outputs Cartesian velocity.

2.2 RL for High Precision Assembly

RL has been studied actively in the area of high precision assembly as it can reduce human involvement and increase the robustness to uncertainties. Inout el al. [19] used a Q-learning based method with LSTM [20] for Q-function approximation to solve low-tolerance peg-in-hole tasks. Luo el at. [21] extended a model-based approach MDGPS [22] with haptic feedback for learning the insertion of a peg into a deformable hole. Fan el at. [23] combined DDPG [24] and GPS [25] to take advantage of both model-free and model-based RL [26] to solve high-precision Lego insertion tasks. Luo el at. [27] combines iLQG [28] with force/torque information by incorporating an operational space controller to solve a group of high-precision assembly tasks. In our work, we use torque/force and

pose in Cartesian space as observations and 6 DOF Cartesian velocities as actions. This method bypasses the robot dynamics, which are usually inaccurate in simulation.

2.3 Leveraging Experience Replay in RL

Experience replay [29] has been used to improve training efficiency in many RL algorithms, particularly for model-free RL, as it's less sample efficient than model-based RL. The technique has become popular after it was incorporated in the DQN [30] agent playing Atari games. Prioritized experience replay [31] is a further improvement to prioritize transitions so agents can learn from the most "relevant" experiences. Hindsight experience replay [32] stored every transition in the replay buffer not only with the original goal but also with a subset of other goals to acquire a more generalized policy. DDPG from Demonstrations [33] modified DDPG to permanently store a set of human demonstrations in the replay buffer to solve a group of insertion tasks. We extend DDPG from Demonstrations to a distributed framework and propose a set of experience replay structures in the context of distributed RL. Details are discussed in Sec. 3.2.

2.4 Distributed RL

Distributed RL can greatly increase training efficiency of model-free RL. Ape-X [72] disconnects exploration from learning by having multiple actors interact with their own environments and select actions from a shared neural network. D4PG [92], with the Ape-X framework, uses a distributional critic update to achieve a more stable learning signal. There are also a growing number of examples applying the distributed architecture to popular RL algorithms, such as Distributed PPO [11] and Distributed BA3C [93]. Since our action space is continuous, we build our algorithm based on RLlib's [38] implementation of Ape-X DDPG.

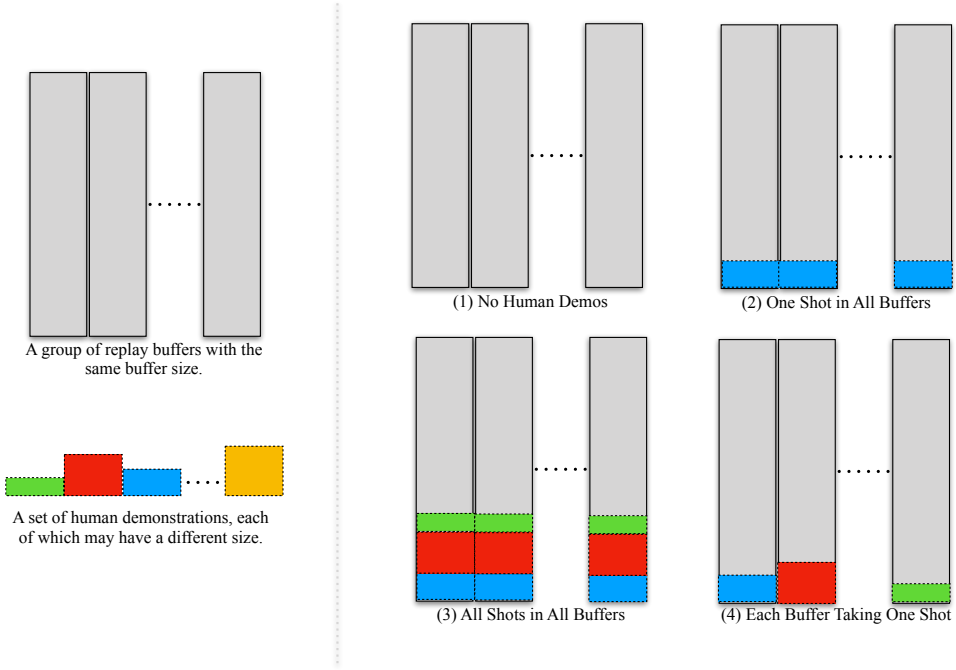


Figure 1: The four types of replay buffer structure for our experiments: (1) No human demonstrations in any buffer; (2) Same one-shot human demonstration in all buffers; (3) All human demonstrations in all buffers; (4) Each buffer with a different one-shot demonstration.

3 Method

As model-free RL algorithms require excessive data, one or multiple shots of human demonstrations are sometimes introduced in the replay buffer R for complex manipulation tasks [33, 94]. However, human demonstrations are not always helpful if the observation space during human demonstration does not match that during training. For example, for the high-precision lap-joint assembly task, we do not have haptic feedback during demonstration in simulation and only visual inspection is used, while during training, force/torque observations are required. Therefore, we propose a novel technique that augments human demonstrations with successful transitions generated by RL agents to improve training efficiency.

3.1 Setup

Observations: The observation space is 13-dimensional. The policy is given as input the position (x, y, z) and orientation (q_x, q_y, q_z, q_w) of the timber piece attached to the robot end-effector, and the torque/force reading $(f_x, f_y, f_z, t_x, t_y, t_z)$ from the sensor, which is mounted on the end of the robot arm. We do not use visual input to simplify the problem.

Actions: The action space is 6-dimensional. The policy outputs the desired linear velocity (v_x, v_y, v_z) and angular velocity (w_x, w_y, w_z) of the timber piece attached to the robot end-effector.

Human demonstrations: For each task, depending on the replay buffer structure (Sec. A), zero, one or six human demonstrations are recorded in simulation, using a game controller to drive the robot end-effector until the joint is successfully assembled. Each demonstration includes all transitions from one successful episode. Each transition is of the form $e_t = (s_t, a_t, s_{t+1}, r_t)$.

Rewards: We use a simple linear reward function based on the distance between the goal pose and the current pose of the timber piece attached to the robot arm for both tasks. Additionally we use a large positive reward (+1000 for the peg-in-hole and +100 for the lap-joint) if the object is within a small distance of the goal pose:

$$r = \begin{cases} -|g - x|, & |g - x| > \epsilon \\ -|g - x| + R, & |g - x| \leq \epsilon \end{cases}$$

where x is the current pose of the object, g is the goal pose, ϵ is a distance threshold, and R is the large positive reward. We use negative distance as our reward function to discourage the behavior of loitering around the goal because the negative distance also contains time penalty.

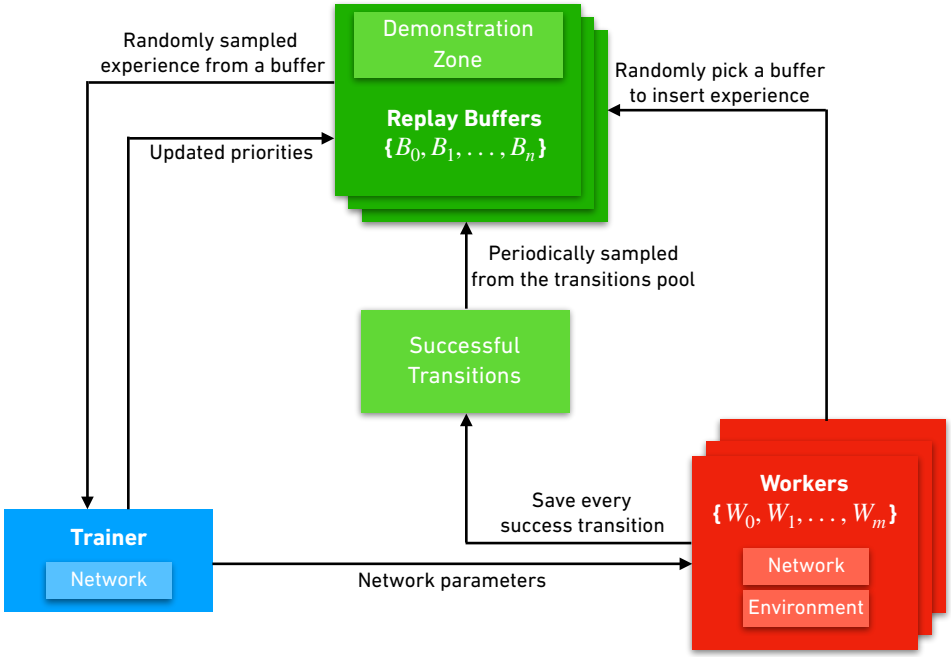


Figure 2: The Dynamic Experience Replay framework: multiple workers, each with its own instance of environment, and multiple replay buffers, each with capacity $\mathbb{C}$ for demonstrations. Human demonstration(s) are stored in the demonstration zones before training starts. During training, all successful transitions that are generated by workers are saved in a pool, which is sampled periodically by each replay buffer and stored in the demonstration zone.

3.2 Replay Structures in Distributed RL

Off-policy RL algorithms perform experience replay by sampling minibatches from a pool of stored samples, which allows the use of arbitrary data like human demonstrations. Based on prioritized experience replay and Ape-X DDPG, we suggest four different replay buffer structures that can take advantage of demonstrations in distributed RL, as shown in Fig. 1.

Each buffer structure consists of a fixed number of replay buffers that load zero, one, or multiple human demonstrations before training starts. Without Dynamic Experience Replay, each buffer permanently keeps all the demonstrations with top priorities during training. The following section discusses how the buffer structures work with Dynamic

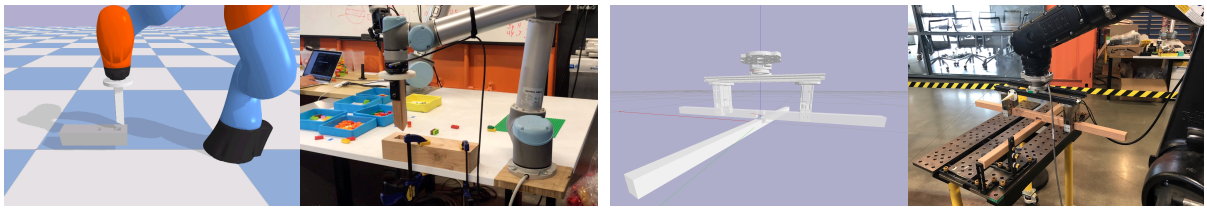
Experience Replay.

3.3 Dynamic Experience Replay

The idea behind Dynamic Experience Replay (DER) is to augment human demonstrations using successful trajectories generated by RL agents during training, especially in cases where human demonstrations are not very helpful. We define *demonstrations* as either human demonstrations or the successful trajectories generated by RL agents. If DER is activated, regardless of the buffer structures mentioned above, each buffer allocates capacity $\mathbb{C}$ specifically for demonstrations. We refer to this as the *demonstration zone*. During training, all the successful episodes generated by RL agents are stored in a pool. Periodically, each replay buffer randomly samples one successful episode from the pool and stores it in the demonstration zone. When the demonstration zone is full, the oldest transitions are discarded. DER's framework in a distributed architecture is shown in Fig. 2.

As in Ape-X, the DER algorithm consists of two concurrent parts, which are workers and a trainer. For each worker, after collecting the transitions of one episode, it randomly chooses a replay buffer and sends over the transitions. The trainer, in parallel, randomly selects a replay buffer and samples a batch of transitions for network update. The trainer also updates the priority of transitions in the selected buffer at the end of the training cycle. See Alg. 4 for a formal description of the algorithm.

A hyper-parameter to experiment with DER is which replay buffer structure to use. In the next section, we compare the four types of replay buffer structure discussed in Sec. 3.2 and how DER affects them.



(a) (b)

Figure 3: Two joint assembly tasks for algorithm evaluation: (a) chamfered peg-in-hole, (b) lap-joint. For both joints, the CAD model used in simulation is used to fabricate the real-world pieces.

4 Experiments

This section is organized as follows. In Sec. 4.1 we introduce distributed RL environments we use for the experiments as well as our training setup and procedure. In Sec. 4.2 we compare the performance of different replay buffer structures with and without DER. In Sec. 4.3 we describe the deployment on the physical robot. The video presenting our experiments is available at `https://sites.google.com/site/dynamicexperiencereplay`.

4.1 Environments

We modeled our assembly tasks in the PyBullet [95] simulation engine. Specifically, we customized two tasks, chamfered peg-in-hole and lap-joint, which correspond to the real-world setup, as shown in Fig. 3.

For the peg-in-hole task, we used a KUKA LRB iiwa robotic arm in simulation and attached a torque/force sensor between the end of the arm and a peg, as shown in Fig. 3(a). In order to be robot agnostic, we limit both the observations and actions in the Cartesian space. This way the trained model can be deployed on any arbitrary robotic arm. To demonstrate the point, we created a robot-less mode for training in simulation for the lap-joint task, as shown in Fig. 3(b). The robot-less setup helps us bypass needing

a robot model in simulation, as most of them are inaccurate.

For each task, we initialized 6 replay buffers and collected 6 human demonstrations. Each demonstration consists of a sequence of transitions of different lengths. Depending on which replay buffer structure is activated, the human demonstration data are used differently, as described in Fig. 1. Training is performed using the Ape-X DDPG algorithm and we adapted it from RLlib's implementation.

Initial states: For the peg-in-hole tasks, the initial angle along the z-axis of the peg is randomized from 0 to 360 degree and other parts are fixed. For the lap-joint tasks, we randomized the initial angle along the z-axis and x-y position of the timber on the ground within a small range. The details of the initial randomnesses are documented in Fig. 4 and Fig. 5.

4.2 Results

In order to evaluate how DER affects the performance we evaluate Ape-X DDPG with and without DER on both two tasks. For each task, we conducted eight types of experiments, which are of four different replay-buffer structures with and without DER. Each experiment was performed on an Amazon AWS c5n.9xlarge instance.

Fig. 4 shows that DER significantly improves the performance of the peg-in-hole task in most of the buffer structures, including No Human Demos, One Shot in All Buffers, and All Shots in All Buffers. For the latter two buffer structures, the average successful rates of DER are greatly higher than vanilla Ape-X DDPG. For the No-Human-Demos buffer structure, although both algorithms have similar average successful rates by the end of the training, DER is nearly two times as fast at achieving the success rate as vanilla Ape-X DDPG. For the Each-Buffer-Taking-One-Shot buffer structure, with DER and without have similar performance.

The lap-joint task is more challenging because the timber pieces have straight corners (no chamfer) and tight tolerance (1mm). Hence, as seen in Fig. 5, the average success rate of each iteration across different training runs is slightly lower than the peg-in-hole task. Fig. 5 shows that DER has better performances than vanilla Ape-X DDPG with two of the buffer structures, No Human Demos and All Shots in All Buffers, while the performances of with DER with the other two buffer structures are similar to without DER. It is unclear why DER does not improve the performance with the Each-Buffer-Taking-One-Shot structure in either task. Further studies need to be conducted.

4.3 Deployment on a physical robot

After training purely in simulation, we deployed the learned policy of the lap-joint task on a KUKA KR60 industrial robot arm, as shown in Fig. 3(b). Our hardware setup includes an ATI Delta 6-axis force/torque sensor, two Schunk parallel-jaw grippers, and two pre-fabricated timber pieces with a half notch on each. As discussed in Sec. 4.1, the observations are force/torque values obtained from the force/torque sensor and pose information of the top timber piece obtained from the robot controller. The actions, linear and angular velocity of the top timber piece, are sent to the robot controller from the policy. The No-Human-Demos buffer structure was used for training the policy, which was successfully deployed on the real robot 3 out of 3 times. We have included the deployment in the video.

5 Discussion and Future Work

This paper proposed a novel technique called Dynamic Experience Replay (DER), which improves training efficiency of an off-policy RL algorithm. The technique uses successful episodes generated by RL agents as demonstrations in replay buffers to aug-

ment human demonstrations. DER can be seen as a technique of over-sampling the under-represented class from imbalanced data in supervised learning. Our technique can be considered as an add-on feature to an arbitrary off-policy RL algorithm and we experimentally demonstrated that with Ape-X DDPG.

We showed that DER in both the peg-in-hole and the lap-joint tasks improved training efficiency in comparison to the vanilla Ape-X DDPG algorithm. For occasions where the vanilla RL algorithm failed to solve the task within the given timeframe, DER could either achieve the training goal or largely improve the success rate. We also showed that the learned policy for the lap-joint task can be successfully deployed on the real robot.

In the future, we would like to evaluate DER on a group of model-free off-policy RL algorithms, such as PPO, and on other assembly tasks. We would also further study DER in terms of hyperparameters, such as the sampling rate and the number of replay buffers.

Algorithm 4: Dynamic Experience Replay

Given:

- a distributed off-policy RL algorithm $\mathbb{A}$,
- an experience replay structure $\mathbb{S}$ [1].
- one-shot or a group of human demonstrations $\mathbb{D}$ (optional)

Initialize $\mathbb{A}$ ▷ Initialize neural networks
Initialize replay buffers $\mathbb{B}$ ▷ Initialize a group of replay buffers
Load $\mathbb{D}$ to $\mathbb{B}$ based on $\mathbb{S}$ ▷ Load human demos to the replay buffers based on the replay structure
Initialize $\mathbb{T}$ ▷ Initialize a pool to save success transitions from agents
For each worker:
for *episode* = 1, **M do**
  $\theta_0 \leftarrow$ Trainer.parameters() ▷ Update the latest network parameters for the trainer
  $s_0 \leftarrow$ Environment.reset() ▷ Get initial state from its own environment
  for $t = 1$, **T do**
    $a_{t-1} \leftarrow \pi_{\theta_{t-1}}(s_{t-1})$ ▷ Choose an action from the current policy
    $(r_{t-1}, s_t) \leftarrow$ Environment.step(a_{t-1}) ▷ Apply the action to the environment
    Transitions.Add($[s_t, a_{t-1}, r_{t-1}, s_{t-1}]$) ▷ Add data to a temp buffer
  $\mathbb{B}_n$.Add(Transitions) ▷ Send the transitions to a randomly selected replay buffer
  if *episode*$_t$ *succeeds* **then**
    ▷ Save success transitions
    $\mathbb{T}$.Add(Transitions)
  Periodically($\theta_t \leftarrow$ Trainer.Parameters()) ▷ Update to the latest network parameters
For the trainer:
for *episode* = 1, **M do**
  $\theta_0 \leftarrow$ InitializeNetwork() **for** $t = 1$, **T do**
    $\tau \leftarrow \mathbb{B}_n$.Sample() ▷ Sample a batch of transitions from a randomly selected buffer
    $l_t \leftarrow$ ComputeLoss($\tau; \theta_t$) ▷ Calculate loss using an off-policy algorithm, like DDPG
    $\theta_{t+1} \leftarrow$ UpdateParameters($l_t; \theta_t$)
    $p \leftarrow$ ComputePriorities() ▷ Calculate priorities of the transitions in buffers[2]
    $\mathbb{B}_n$.SetPriority(p) ▷ Update priorities to the selected buffer
  Periodically($\mathbb{B}_i$.Update($\mathbb{T}_j$)) ▷ Replace previous demos with a success transition from the pool

1. The four different experience replay structures are discussed in Sec 3.2.
2. We use absolute TD error for the calculation.

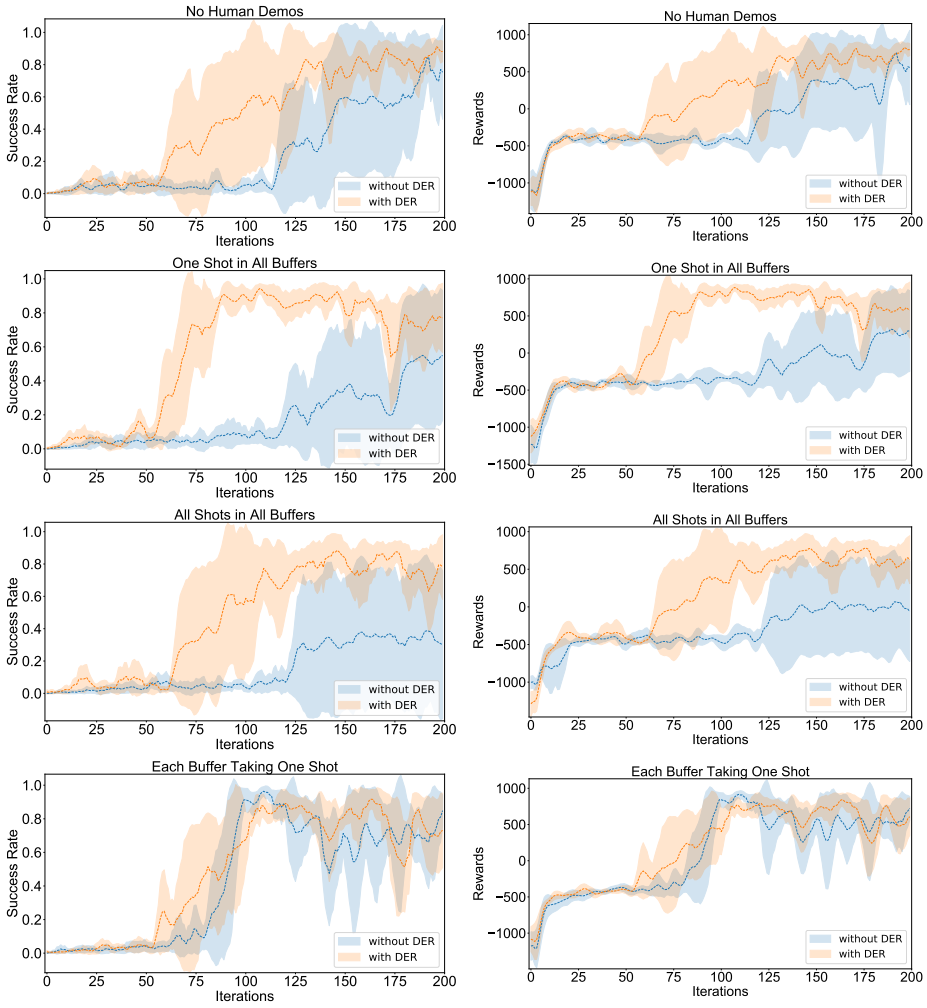


Figure 4: Success rate comparison (left) and reward comparison (right) for the peg-in-hole experiments, in which the initial angle of the peg along the z-axis is randomized within [0, 360°]. Each plot compares the performances of a replay buffer structure with and without DER. Each iteration consists of 50 to 80 episodes and is approximately 200,000 timesteps. The dotted lines show the mean of each iteration across 3 trainings with different random seeds and the shaded areas show the 95% confidence bound. Each training experiment is terminated at 200 iterations.

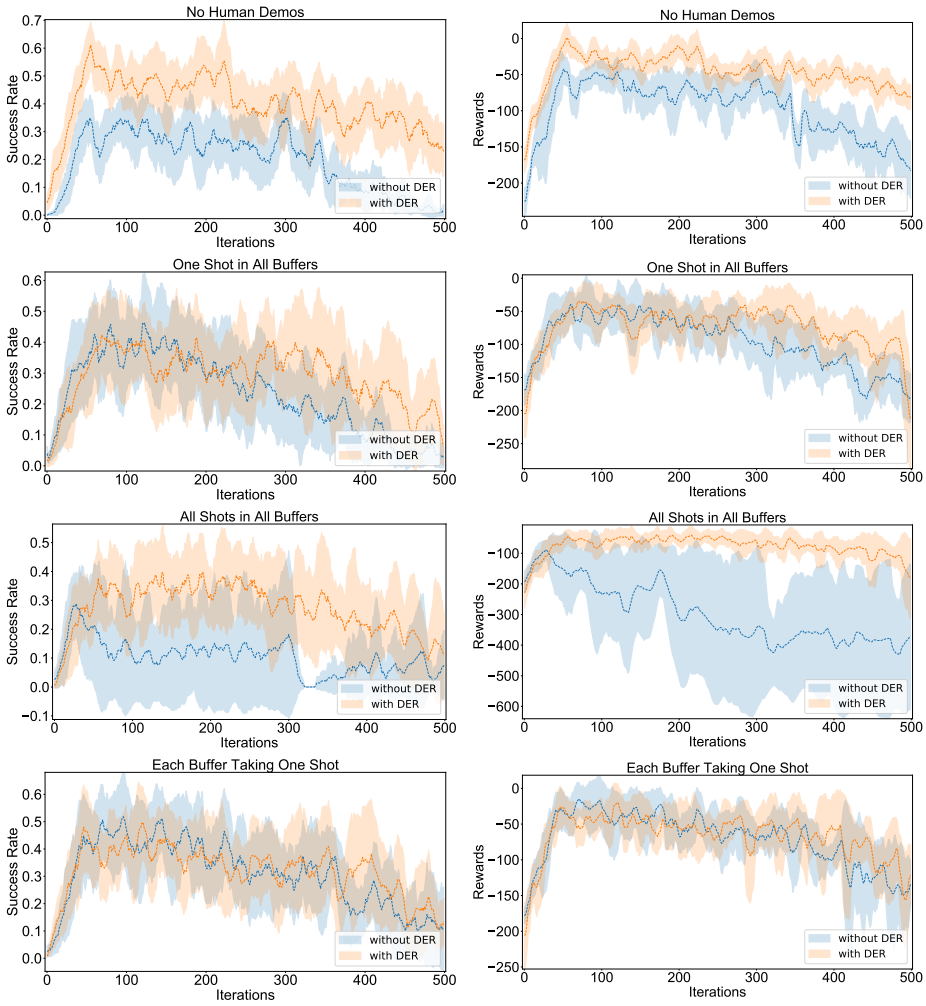


Figure 5: Success rate comparison (left) and reward comparison (right) for the lap-joint experiments, in which the initial angle of the ground timber piece along the z-axis is randomized within $[-2°, 0]$ and the initial position [-2mm, 2mm] in both x and y. Each graph compares the performances of a replay buffer structure with and without DER. Each iteration consists of 50 to 80 episodes and is approximately 200,000 timesteps. The dotted lines show the mean of each iteration across 3 trainings with different random seeds and the shaded areas show the 95% confidence bound. Each training experiment is terminated at 500 iterations.

Appendix B

Thoughts of *Machine Learning with Visual Overlay*

Andreas Schlegel, 2019

The work itself can be seen as a commentary on current developments in computation. Computer generated images present an ever-growing challenge to our perception of reality due to hyper-realistic renderings that can often be observed in special effects in movies, computer games and XR-applications. Artificial Intelligence (AI) techniques such as StyleGAN or DeepFakes produce images surprisingly indistinguishable from a real images taken by a camera our way to capture, share and remember moments in time visually a development we will find progressively challenging within social, cultural and ethical context. Equally, technologies and machines related to AI and Machine Learning (ML) are increasingly seen as a threat to our privacy and as a competition to human knowledge, skills and labor, especially where automation and robots can be more accurate and enduring.

It should be emphasized that, in contrast to the images resulting from for example

convolutional neural networks such as DeepDream renderings, the visuals rendered here (Program A) are only indirectly a result of the RL program (Program B) or rather arise from the decisions and actions taken by the learning algorithm.

As I am more familiar with programming paradigms like imperative, procedural or object-oriented programming , this work, in particular Program B, allows me to experiment, play and develop an understanding of a technology (RL) that sees rapid development and application in many aspects of daily life through learning algorithms and systems.

The emergence of self-learning systems opens up new frontiers not only in computing, for which we can expect developments that are currently difficult or impossible to realize with conventional techniques. Some of the developments I look forward to are:

- Learning systems to accel in areas such as health-care, fintech, automation, robotics
- Machine-automated processes to replace repetitive tasks carried out by humans; humans will have to redefine many aspects of work where creativity will have to replace repetition
- Humans to redefine what challenge, especially in games, means in an era that produces self-learning systems which will continue to beat humans in mostly any board or computer game
- Lesser expert programming-knowledge required when training self-learning systems; everyone will be able to train such a system for individual or general use, needs, purposes
- Computational applications and interfaces to become more intuitive and adaptive
- Interactions between machines and humans to become more conversational rather than instructional

- Machines to develop behavior

To conclude, both programs are very different in their approach and in what they stand for. The key aspect for me is the potential of ML and especially RL and the underlying paradigm shift symbolized in the presentation of Program A. The results embodied by Program B are more of a personal experience and reflect my passion for abstract and unusual images and the complexity they contain, which is influenced by the actions and interaction of the other program.

Appendix C

Recognition of RL5

An earlier version of RL5 was mentioned by OpenAI at its first hackathon event. OpenAI is a leading artificial intelligence research center based in San Francisco. It is products consisting of OpenAI Gym, a benchmark of RL environments to evaluate RL algorithms, OpenAI Five, the first AI system to beat the world champions in an esports game, and Dactyl, a human-like robot hand to manipulate physical objects with unprecedented dexterity.

The author was accepted by OpenAI to attend a selective hackathon with 100 members in the artificial intelligence community. The event had over 500 applications within two days of announcing the event. RL5 was selected as one of seven their favorites projects and was featured on their official blog [1].

[1] `https://openai.com/blog/hackathon-follow-up/`

Bibliography

[1] H. Cohen, *The further exploits of aaron, painter*, *Stanford Humanities Review* **4** (1995), no. 2 141–158.

[2] A. Mordvintsev, C. Olah, and M. Tyka, *Deepdream-a code example for visualizing neural networks*, *Google Research* **2** (2015), no. 5.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[4] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et. al.*, *Learning representations by back-propagating errors*, *Cognitive modeling* **5** (1988), no. 3 1.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of go with deep neural networks and tree search*, *Nature* **529** (2016) 484–503.

[6] A. Maekawa, A. Kume, H. Yoshida, J. Hatori, J. Naradowsky, and S. Saito, *Improvised robotic design with found objects*, .

[7] R. Saunders and P. Gemeinboeck, *Performative body mapping: A creative robotics method for learning expressive movement*, in *Proc. of NIPS 2018, Workshop on Machine Learning for Creativity and Design*, vol. 8, 2018.

[8] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, *Image-to-image translation with conditional adversarial networks*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[9] D. Ha and D. Eck, *A neural representation of sketch drawings*, *arXiv preprint arXiv:1704.03477* (2017).

[10] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et. al.*, *Learning dexterous in-hand manipulation*, *arXiv preprint arXiv:1808.00177* (2018).

[11] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller, *et. al.*, *Emergence of locomotion behaviours in rich environments*, *arXiv preprint arXiv:1707.02286* (2017).

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, *arXiv preprint arXiv:1509.02971* (2015).

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, *arXiv preprint arXiv:1606.01540* (2016).

[14] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et. al.*, *Mastering the game of go without human knowledge*, *Nature* **550** (2017), no. 7676 354.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, *arXiv preprint arXiv:1312.5602* (2013).

[16] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dbiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Jzefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, *Dota 2 with large scale deep reinforcement learning*, *arXiv preprint* (2019).

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, *arXiv preprint arXiv:1707.06347* (2017).

[18] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et. al.*, *Grandmaster level in starcraft ii using multi-agent reinforcement learning*, *Nature* (2019) 1–5.

[19] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, *Deep reinforcement learning for high precision assembly tasks*, in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 819–825, IEEE, 2017.

[20] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, *Neural computation* **9** (1997), no. 8 1735–1780.

[21] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino, *Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects*, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2062–2069, IEEE, 2018.

[22] W. H. Montgomery and S. Levine, *Guided policy search via approximate mirror descent*, in *Advances in Neural Information Processing Systems*, pp. 4008–4016, 2016.

[23] Y. Fan, J. Luo, and M. Tomizuka, *A learning framework for high precision industrial assembly*, *arXiv preprint arXiv:1809.08548v3* (2018).

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, in *6th International Conference on Learning Representations*, 2016.

[25] S. Levine and V. Koltun, *Guided policy search*, in *International Conference on Machine Learning*, pp. 1–9, 2013.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, *Reinforcement learning on variable impedance controller for high-precision robotic assembly*, *arXiv preprint arXiv:1903.01066* (2019).

[28] E. Todorov and W. Li, *A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems*, in *Proceedings of the 2005, American Control Conference, 2005.*, pp. 300–306, IEEE, 2005.

[29] L.-J. Lin, *Self-improving reactive agents based on reinforcement learning, planning and teaching*, *Machine learning* **8** (1992), no. 3-4 293–321.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et. al.*, *Human-level control through deep reinforcement learning*, *Nature 518, pages 529533* (2015).

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, *Prioritized experience replay*, *arXiv preprint arXiv:1511.05952* (2015).

[32] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, *Hindsight experience replay*, in *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

[33] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, *Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards*, *arXiv preprint arXiv:1707.08817* (2017).

[34] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." `https://github.com/openai/baselines`, 2017.

[35] M. Schaarschmidt, A. Kuhnle, B. Ellis, K. Fricke, F. Gessert, and E. Yoneki, *LIFT: Reinforcement learning in computer systems by learning from demonstrations*, *CoRR* **abs/1808.07903** (2018) [arXiv:1808.0790].

[36] M. Plappert, "keras-rl." `https://github.com/keras-rl/keras-rl`, 2016.

[37] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." `https://github.com/hill-a/stable-baselines`, 2018.

[38] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, *Rllib: Abstractions for distributed reinforcement learning*, *arXiv preprint arXiv:1712.09381* (2017).

[39] A. Stooke and P. Abbeel, *rlpyt: A research code base for deep reinforcement learning in pytorch*, *arXiv preprint arXiv:1909.01500* (2019).

[40] E. Todorov, T. Erez, and Y. Tassa, *Mujoco: A physics engine for model-based control*, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[41] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, *The arcade learning environment: An evaluation platform for general agents*, *Journal of Artificial Intelligence Research* **47** (jun, 2013) 253–279.

[42] N. Bhonker, S. Rozenberg, and I. Hubara, *Playing snes in the retro learning environment*, *arXiv preprint arXiv:1611.02205* (2016).

[43] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg, *Ai safety gridworlds*, *arXiv preprint arXiv:1711.09883* (2017).

[44] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, *Habitat: A Platform for Embodied AI Research*, *arXiv preprint arXiv:1904.01201* (2019).

[45] S. Shah, D. Dey, C. Lovett, and A. Kapoor, *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*, in *Field and Service Robotics*, 2017. 1705.0506.

[46] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, *Matterport3d: Learning from rgb-d data in indoor environments*, *arXiv preprint arXiv:1709.06158* (2017).

[47] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, *Gibson env: Real-world perception for embodied agents*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9068–9079, 2018.

[48] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, *Unity: A general platform for intelligent agents*, *arXiv preprint arXiv:1809.02627* (2018).

[49] P. Galanter, *What is generative art? complexity theory as a context for art theory*, in *In GA2003–6th Generative Art Conference*, Citeseer, 2003.

[50] I. Xenakis, *Formalized music: thought and mathematics in composition*. No. 6. Pendragon Press, 1992.

[51] F. Nake, *The disappearing masterpiece: Digital image & algorithmic revolution*, in *XCoAX 2016: Proceedings of the Fourth Conference on Computation, Communication, Aesthetics, and X*, pp. 12–27, 2016.

[52] P. Prince, *Artists and computers: A retrospective*, *IEEE Computer Graphics and Applications* (1986), no. 8 8–11.

[53] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, *Semi-supervised learning with deep generative models*, in *Advances in neural information processing systems*, pp. 3581–3589, 2014.

[54] M. Mazzone and A. Elgammal, *Art, creativity, and the potential of artificial intelligence*, in *Arts*, vol. 8, p. 26, Multidisciplinary Digital Publishing Institute, 2019.

[55] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[56] T. Karras, S. Laine, and T. Aila, *A style-based generator architecture for generative adversarial networks*, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

[57] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, *et. al.*, *Deep speech: Scaling up end-to-end speech recognition*, *arXiv preprint arXiv:1412.5567* (2014).

[58] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[59] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, *Exploring the limits of language modeling*, *arXiv preprint arXiv:1602.02410* (2016).

[60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[61] L. A. Gatys, A. S. Ecker, and M. Bethge, *A neural algorithm of artistic style*, *arXiv preprint arXiv:1508.06576* (2015).

[62] L. McCarthy, C. Reas, and B. Fry, *Getting Started with P5. js: Making Interactive Graphics in JavaScript and Processing*. Maker Media, Inc., 2015.

[63] D. Shiffman, *The Nature of Code: Simulating Natural Systems with Processing*. 2012.

[64] N. ITP, "Ml5js: Friendly machine learning for the web." `https://ml5js.org/`, 2018.

[65] A. Roberts, C. Hawthorne, and I. Simon, *Magenta. js: A javascript api for augmenting creativity with deep learning*, .

[66] K. Pearson, *The problem of the random walk*, *Nature* **72** (1905), no. 1867 342.

[67] J. Conway, *The game of life*, *Scientific American* **223** (1970), no. 4 4.

[68] T. A. Witten and L. M. Sander, *Diffusion-limited aggregation*, *Physical Review B* **27** (1983), no. 9 5686.

[69] J. Luo, *Abstract reality*, in *SIGGRAPH Asia 2017 Art Gallery*, p. 2, ACM, 2017.

[70] J. Luo and W. Zhang, *Lavin*, in *ACM SIGGRAPH 2019 Art Gallery*, p. 17, ACM, 2019.

[71] K. Simonyan, A. Vedaldi, and A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, *arXiv preprint arXiv:1312.6034* (2013).

[72] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, *Distributed prioritized experience replay*, *arXiv preprint arXiv:1803.00933* (2018).

[73] J. Luo, S. Green, P. Feghali, G. Legrady, and Ç. K. Koç, *Reinforcement learning and trustworthy autonomy*, in *Cyber-Physical Systems Security*, pp. 191–217. Springer, 2018.

[74] J. Luo, S. Green, P. Feghali, G. Legrady, and C. K. Koç, *Visual diagnostics for deep reinforcement learning policy development*, *arXiv preprint arXiv:1809.06781* (2018).

[75] C. J. Watkins and P. Dayan, *Q-learning*, *Machine learning* **8** (1992), no. 3-4 279–292.

[76] R. J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, *Machine learning* **8** (1992), no. 3-4 229–256.

[77] R. Bellman, *An introduction to the theory of dynamic programming*, tech. rep., RAND CORP SANTA MONICA CA, 1953.

[78] A. G. Barto, R. S. Sutton, and C. W. Anderson, *Neuronlike adaptive elements that can solve difficult learning control problems*, *IEEE transactions on systems, man, and cybernetics* (1983), no. 5 834–846.

[79] J. F. Schaefer, *On the control of unstable mechanical systems*, in *Proc. of the Third Congress of IFAC*, vol. 1, pp. 6c–1, 1966.

[80] K. Furuta, M. Yamakita, and S. Kobayashi, *Swing-up control of inverted pendulum using pseudo-state feedback*, *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* **206** (1992), no. 4 263–269.

[81] T. Shinbrot, C. Grebogi, J. Wisdom, and J. A. Yorke, *Chaos in a double pendulum*, *American Journal of Physics* **60** (1992), no. 6 491–499.

[82] S. Franklin and A. Graesser, *Is it an agent, or just a program?: A taxonomy for autonomous agents*, in *International Workshop on Agent Theories, Architectures, and Languages*, pp. 21–35, Springer, 1996.

[83] C. W. Reynolds, *Steering behaviors for autonomous characters*, in *Game developers conference*, vol. 1999, pp. 763–782, Citeseer, 1999.

[84] J. Luo and H. Li, *Dynamic experience replay*, *2019 Conference on Robot Learning* (2019).

[85] J. Rojas, S. Coros, and L. Kavan, *Deep reinforcement learning for 2d soft body locomotion*, .

[86] J. F. Mellor, E. Park, Y. Ganin, I. Babuschkin, T. Kulkarni, D. Rosenbaum, A. Ballard, T. Weber, O. Vinyals, and S. Eslami, *Unsupervised doodling and painting with improved spiral*, *arXiv preprint arXiv:1910.01007* (2019).

[87] Y. Ganin, T. Kulkarni, I. Babuschkin, S. Eslami, and O. Vinyals, *Synthesizing programs for images using reinforced adversarial learning*, *arXiv preprint arXiv:1804.01118* (2018).

[88] J. Kober, J. A. Bagnell, and J. Peters, *Reinforcement learning in robotics: A survey*, *The International Journal of Robotics Research 32.11 (2013), pp. 12381274* (2013).

[89] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization*, in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[90] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, *Smote: synthetic minority over-sampling technique*, *Journal of artificial intelligence research* **16** (2002) 321–357.

[91] H. He, Y. Bai, E. A. Garcia, and S. Li, *Adasyn: Adaptive synthetic sampling approach for imbalanced learning*, in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1322–1328, IEEE, 2008.

[92] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap, *Distributed distributional deterministic policy gradients*, in *6th International Conference on Learning Representations*, 2018.

[93] I. Adamski, R. Adamski, T. Grel, A. Jedrych, K. Kaczmarek, and H. Michalewski, *Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes*, in *International Conference on High Performance Computing*, pp. 370–388, Springer, 2018.

[94] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, *Overcoming exploration in reinforcement learning with demonstrations*, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, IEEE, 2018.

[95] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, *GitHub repository* (2016).